;login:

The USENIX Association Newsletter

Volume 11, Number 2 **March/April 1986**

CONTENTS

Executive Director's Report 3
Candidates for Officers and Directors of the USENIX Association 4
GKS Code Being Distributed as First 1986 Software Tape 17
Announcing the Fourth Printing of USENIX 4.2BSD Manuals 17
The Summer 1986 USENIX Conference 19
Tutorials to be Presented at the Summer '86 USENIX Conference 20
Future Meetings 28
On Text 29
A Report on the Accuracy of Some Floating Point Math Functions on Selected Computers 31
Eugene H. Spafford and John C. Flaspohler
Local User Groups 57
Publications Available 58
4.2BSD Manual Reproduction Authorization and Order Form 59

The closing date for submissions for the next issue of *;login:* is April 23, 1986

NOTICE

;login: is the official newsletter of the USENIX Association, and is sent free of charge to all members of the Association.

The USENIX Association is an organization of AT&T licensees, sub-licensees, and other persons formed for the purpose of exchanging information and ideas about UNIX† and UNIX-like operating systems and the C programming language. It is a non-profit corporation incorporated under the laws of the State of Delaware. The officers of the Association are:

President	Alan G. Nemeth
Vice-President	Deborah K. Scherrer
Secretary	Lewis A. Law
Treasurer	Waldo M. Wedel
Directors	Thomas Ferrin
	Steve C. Johnson
	Lou Katz
	Michael D. Tilson
Executive Director	James E. Ferguson

The editorial staff of *;login:* is:

Editor and Publisher	James E. Ferguson usenix!jim
Copy Editor	Michelle Peetz {masscomp,usenix}!mmp
Humor Editor	Rob Kolstad {convex,usenix}!kolstad
Production Editor	Tom Strong usenix!tom
Editorial Director	Lou Katz {ucbvax,usenix}!lou

Other staff members are:

Betty J. Madden Office Manager
Connie I. Howard Membership Secretary

USENIX Association Office
P.O. Box 7
El Cerrito, CA 94530
(415) 528-8649
{ucbvax,decvax}!usenix!office

Judy DesHarnais Conference Coordinator

USENIX Conference Office
P.O. Box 385
Sunset Beach, CA 90742
(213) 592-3243 or 592-1381
{ucbvax,decvax}!usenix!judy

John L. Donnelly Exhibit Manager

USENIX Exhibit Office
Oak Bay Building
4750 Table Mesa Drive
Boulder, CO 80303
(303) 499-2600
{ucbvax,decvax}!usenix!johnd

Contributions Solicited

Members of the UNIX community are heartily encouraged to contribute articles and suggestions for *;login:*. Your contributions may be sent to the editors electronically at the addresses above or through the U.S. mail to the Association office. The USENIX Association reserves the right to edit submitted material.

;login: is produced on UNIX systems using *troff* and a variation of the –me macros. We appreciate receiving your contributions in *n/troff* input format, using any macro package. If you contribute hardcopy articles please leave left and right margins of 1″ and a top margin of 1½″ and a bottom margin of 1¼″. Hardcopy output from a line printer or most dot-matrix printers is not reproducible.

Acknowledgments

The Association uses a VAX‡ 11/730 donated by the Digital Equipment Corporation for support of office and membership functions, preparation of *;login:*, and other association activities. It runs 4.2BSD, which was contributed, installed, and is maintained by mt Xinu. The VAX uses a sixteen line VMZ-32 terminal multiplexor donated by Able Computer of Irvine, California.

Connected to the VAX is a QMS Lasergrafix* 800 Printer System donated by Quality Micro Systems of Mobile, Alabama. It is used for general printing and draft production of *;login:* with *ditroff* software provided by mt Xinu.

†UNIX is a trademark of AT&T Bell Laboratories.

‡VAX is a trademark of Digital Equipment Corporation.
*Lasergrafix is a trademark of Quality Micro Systems.

Executive Director's Report

(a/k/a Jim's Verbiage)

Board Election

The pictures, background information, and position statements of the candidates for officers and directors of USENIX Association for the 1986-88 term are on the following 13 pages. We encourage you to read their qualifications and opinions carefully. It will not be an easy choice as all of the candidates are well qualified and would make good officers and directors.

Ballots will be mailed about April 11, and they *must be received* by the Association Office on or before May 9, 1986, to be counted. Ballots will be mailed only to those members who have paid their 1986 membership dues.

USENIX belongs to you, the members. Please take the time and interest to vote for the candidates who will guide USENIX in the direction you want it to go during the next two years.

Denver Conference – January 1986

The USENIX 1986 Winter Conference in Denver was an overwhelming success. With the change in meeting format and being only two weeks before UniForum 1986 in Anaheim, we were hoping for an attendance of 500. Instead there were 1,157 attendees, plus more favorable comments than at any prior winter conference. The format of running three 1-day symposia or workshops concurrent with tutorials was so well liked, it will be repeated at the 1987 Winter Conference in Washington, DC.

Atlanta Conference – June 1986

Nearly all of the arrangements are complete for the USENIX 1986 Summer Conference in Atlanta, June 9-13. In response to popular demand, there will be two days of tutorials before the conference this time – with the five most popular tutorials being offered twice, both on Monday and Tuesday. Descriptions of the tutorials being offered are printed in this issue of *;login:*, as well as in the pre-registration packet which will be mailed in April. Please mark your calendars and plan to attend the Atlanta Conference, June 9-13.

Washington Conference – January 1987

The responses to our 1986 Membership Survey revealed the majority of the members prefer our Winter Conferences to be held in the same city at the same time as /usr/group's UniForum trade shows. The Boards of Directors of both USENIX Association and /usr/group have worked long and hard to resolve the differences and misunderstandings between the two organizations.

We are pleased to announce the 1987 USENIX Winter Conference will be held in Washington, DC, at the Shoreham Hotel on January 20-23, 1987 – the same time /usr/group's UniForum 1987 is being held at the Washington Convention Center.

Membership and Dues Billings

We thank all of you who promptly paid your 1986 membership dues and completed the membership survey. Only three months into 1986, we already have more paid members than we had after twelve months in 1985. To the 470 new members who joined via the Denver Conference, we welcome you and thank you for joining *USENIX – the professional and technical UNIX™ association.*

Jim Ferguson
Executive Director

Candidates for Officers and Directors of the USENIX Association

Alan G. Nemeth

CANDIDATE FOR PRESIDENT

Background Information

Alan Gary Nemeth is a corporate consultant in the area of operating systems for Prime Computer, Inc., where he has broad responsibility for overall technical and strategic direction of many Prime products. He came to Prime from Bolt, Beranek and Newman, Inc., where he was responsible for developing BBN's UNIX activities, including the C machine. Previously he worked at the MIT Lincoln Laboratory on simulations of the air traffic system, graphics, and networking. Dr. Nemeth holds an S.B. in Applied Mathematics from Columbia University, and an M.S. and Ph.D. in Applied Mathematics from Harvard University. Dr. Nemeth has been president of the USENIX Association since June, 1984. Previously he was a director of the Association for two years and was the technical chairman of the July 1982 USENIX conference. Dr. Nemeth was a member of the DARPA Steering Committee for the Computer Science Research Group at U.C. Berkeley, and is a member of ACM and the IEEE Computer group.

Position Statement

I have been a member of the USENIX board for almost four years, and involved with the Association for over ten years. During this period, the Association has grown from a loosely-knit gathering of a few colleagues who could assist each other, through a stage of rapid expansion to the current phase of relative stability. Each of these phases has brought its own opportunities, problems, and crises. I expect the Association to continue to evolve and mature as the community it serves continues to develop.

The purpose of the Association has always been support for the "exchange and communication of research and technological information and ideas pertaining to UNIX and UNIX-related computer systems." There are many forms of this exchange – we have done particularly well at some, and have a distance to go in others.

Our conferences remain our most significant activities. Over the past two years, the attendance has stabilized at about 1,500 attendees, and the quality has steadily improved. Our conference chairmen have handled their roles professionally and worked with their committees to improve the refereeing process. The proceedings has now become an assumed part of our conferences, despite cliff-hanging stories each time one is produced. The tutorial program has also expanded dramatically, reflecting the change in the community. Overall, this area of activity is alive, vital, critical to our success, and supported enthusiastically.

Nonetheless, there are opportunities for improving our conference activities. The recent Denver meeting demonstrated quite successfully an alternate format of one-day focused symposia and tutorials. This worked well, and we will use this format again at our next winter meeting. A critical question is when we should shift from a semi-annual to an annual cycle for the large conferences. I continue to believe in the need for meeting twice a year, but this is subject to change depending on the availability of continuing high-quality technical content.

Another major issue in our conference planning is our relationship with /usr/group and the UniForum meetings. I have continued to champion the idea that we assist the overall UNIX community by holding USENIX meetings in conjunction with UniForum meetings. There has been extensive discussion on the validity of this position for the Association, but I believe that UNIX will continue to be technically interesting and commercially viable as long as the various groups in the community recognize the importance of the different viewpoints represented by USENIX and /usr/group. However, we need to retain our different identities as we work together.

We have tried other communication formats, including small workshops, newsletters, and distribution tapes. All of these have been helpful, but need further development to provide real value to our members. Over the next two years, I would like to focus on developing additional approaches for exchanging information among our members, as well as improving the existing ones.

Lastly, I would like to note our overall success with hiring professional staff to operate the Association. These employees have enabled the Association to provide services in a more reliable fashion. I see a continuing need for the Board to provide direction to our staff, and to establish personnel policies to keep these employees satisfied and motivated.

I would appreciate your support and suggestions.

Deborah K. Scherrer

CANDIDATE FOR VICE-PRESIDENT

Background Information

Deborah K. Scherrer is a Computer Scientist for mt Xinu, where she manages their UNIX Support Services Division. After completing her undergraduate and graduate studies at U.C. Berkeley, she worked for 10 years as a Computer Scientist at Lawrence Berkeley Laboratory, where she managed several UNIX-related projects including one to develop a computing environment for seismic research and nuclear test ban treaty verification. She is a member of the *UNIX Review* Editorial Board, a Contributing Editor for *UNIX/World*, founder of the Software Tools Users Group, and serves on the Board of Directors of the C Users Group. She has been serving on the USENIX Board of Directors since 1980, most recently as Vice-President.

Position Statement

At the time of the 1984 elections, USENIX was enmeshed in determining its role in the UNIX community. The membership was diverse, UNIX commercial interests were expanding at a rapid pace, and the UNIX system itself was taking on new and larger dimensions and variations designed to serve assorted subcommunities. In addition, much important research in computer science was being performed using UNIX as its base. The emergence of other UNIX user groups was pressuring us to further clarify our role and adequately fulfill our functions. Thus USENIX was challenged to maintain its professional, technical orientation in the face of a rapidly changing, expanding, and diversifying community.

To focus on its goal of fostering "the exchange and communication of research and technological information and ideas pertaining to UNIX" (as our Bylaws state), USENIX undertook a series of projects designed to more adequately serve the needs of the technical community. We expanded the conferences to include more in-depth tutorials, and experimented with the single-day, single-topic format used in Denver. We also initiated a series of independent workshops, open to a small number of technical experts in a particular field. In addition, we sponsored several projects designed to fill needs in the technical community, including the 4.2BSD manuals production, the UUCP mapping project, and the Stargate experiment.

The primary activity of our Association has always been our technical conferences. In a very short time we have seen these expand from an informal "sharing of secrets" to professional, high quality affairs. We now have proceedings available at the conference, and have made arrangements for additional publication of the best papers. I expect to see the quality improve even further as new research challenges and expands the existing technical base. But I also expect to see the organization expanding into other services that will prove equally valuable. Our software exchange program needs revision to improve its usefulness. We have already designed and set up a public domain distribution arrangement, and I would also like to see us expand our distribution of licensed material. In addition, we could develop on-line access to an assortment of information databases which could be fed and accessed by the community. I would also like to see USENIX expand its funding of projects in the UUCP area, where I see the greatest need in the community right now. And, finally, our publications should be expanded to include a more technically-oriented bulletin or journal providing high-quality submissions as well as status reports of research projects underway.

I have invested much of my time and energy in this organization because I believe it has much to offer a very special and free-spirited group of people. I enjoy the USENIX community very much and would like to continue serving as Vice-President to help USENIX develop into the top quality, professional, technical society I believe it can be.

Wally Wedel

CANDIDATE FOR SECRETARY

Background Information

I have worked at NBI, Inc. of Boulder, Colorado for the past five years developing advanced office automation systems. Prior to this, I worked at The University of Texas Computation Center for eight years developing and providing a variety of computing services in support of the research and educational goals of the University. I became acquainted with UNIX back in 1974 when the University of Texas acquired a UNIX system for support of research and education in the College of Communications. I have used it extensively ever since. Most recently I have participated in the design and development of UNIX-based Office Automation and Engineering Work Stations now marketed by NBI.

During my term as Treasurer of USENIX, I have made several significant contributions to the Association. I have worked to implement better, more complete accounting procedures. I have seen the USENIX financial position improve to the point where the Association has over $250,000 in reserves. I have taken on board level responsibility for the meeting arrangements of all our conferences.

Position Statement

USENIX has been remarkably successful at conducting high quality, technical meetings for many years now. We have developed a Summer Conference format which provides for reporting of research results along with a solid set of tutorials and a technically focused vendor exhibition. We have been experimenting with the Winter Conference format and seem to have found a good format. An entire day devoted to a single topic allows for an in-depth exploration of the topic with lots of good discussion. Supplementing the symposium topic with good tutorials gives lots of good information flow. We are considering adding a tutorials-only meeting to satisfy the high demand for good quality tutorials.

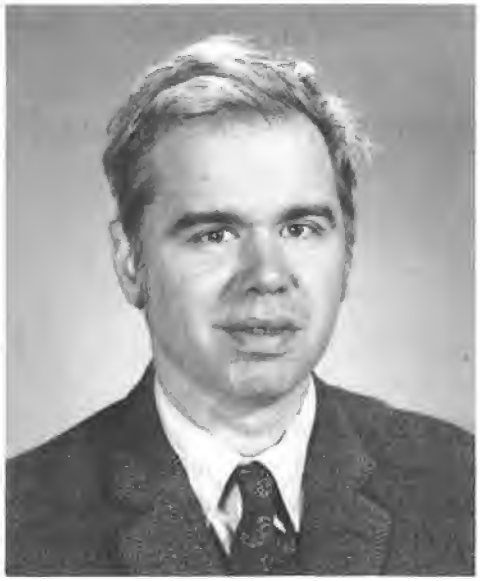
We now need to focus attention on upgrading our newsletter. It needs more articles with solid technical content and a more inviting layout. Now that we have ample financial resources to devote to this effort, we should do so. We should obtain the services of a respected member of the UNIX community as a technical editor. We should also employ a graphics design specialist to help make the journal look like a high quality technical publication. Finally, we need to enlist the cooperation of our membership in contributing high quality technical material for publication.

Our relations with /usr/group have improved over the last two years to a point where it appears USENIX and /usr/group can work together effectively. A spirit of cooperation as well as competition has developed between the groups which should be encouraged. I favor fostering this cooperation as long as we can assure that it is in our best interests.

The USENIX financial position has never been stronger. We should use these resources to develop new, high quality services for our members. The Board has asked our membership to propose projects for the benefit of the entire community which they can't get funded elsewhere. USENIX should stand ready to provide this help to foster new and innovative uses of UNIX.

Steve Johnson

CANDIDATE FOR TREASURER



Background Information

A.B., Haverford College, M.S. and Ph.D. in mathematics from Columbia University. Worked at AT&T (Bell Labs and Information Systems) 1967-1986. Research work included psychometrics, computer algebra, parsing, code generation, complexity theory, compiler construction, and VLSI design. UNIX experience includes the writing of *yacc, pcc, lint,* early versions of *spell,* etc.; with Dennis Ritchie, the port of UNIX to the Interdata 8/32. Was the former head of both the Computer Systems Research Department (Bell Labs) and the Language Development Department (Information Systems). In 1986, named a Bell Laboratories Fellow. Dr. Johnson is currently Director of Programming Languages for The Dana Group in Sunnyvale, California.

Position Statement

There is a place for a technical society dedicated to advancing the evolution of the UNIX operating system. To keep USENIX firmly in that place, we must continue to strengthen the technical content of our conferences, and explore other means of encouraging technical interchanges among our members.

Peter Honeyman

CANDIDATE FOR DIRECTOR

Background Information

B.G.S., University of Michigan, 1975.
M.S.E. and M.A. (EECS) Princeton University, 1979.
Ph.D. (EECS) Princeton University, 1980.
MTS at Bell Labs, 1980-1983.
Assistant Professor at Princeton University, 1983-present.

Research in relational database theory, VLSI, networks, electronic addressing and routing.

Position Statement

The USENIX Association can be a stronger voice for the UNIX technical community. For example, USENIX should address the needs of the membership by acting as the interface to the DARPA Internet. To this end, I propose that we form a task force for UUCP/Internet gatewaying. Within our community, we speak with many voices, and represent many views. Through USENIX, we can reach a consensus to provide gateway services to the members, improve our network, and better represent our needs to the Internet community. In addition, we should establish a relationship with CSNET that accommodates the cross-membership of many USENIX institutional members.

I support a strong annual conference, divorced from UniForum and /usr/group. Although narrow in scope, the USENIX Conferences compare favorably with ACM conferences such as POPL and SOSP; we have traditionally attracted solid systems-oriented work and are now taken seriously in the academic community.

I believe USENIX should conduct a membership drive, not to increase our already bountiful treasury, but to speak with more authority as the representative of the UNIX technical community. Finally, we should maintain an arms-length relationship with both /usr/group and USENET.

Rob Kolstad

CANDIDATE FOR DIRECTOR

Background Information

B.A.Sc. (Computer Science) Southern Methodist University, 1974.
M.S.E.E. (Electrical Engineering) Notre Dame University, 1976.
Ph.D. (Computer Science) University of Illinois at Urbana/Champlain, 1982.

Manager of Operating Systems and Computer Systems Manager, CONVEX Computer Corporation. Co-Program Chairman of 1985 Winter USENIX Conference. Frequent speaker at USENIX conferences. Ten years of UNIX experience.

Position Statement

The success of users' groups in general depends heavily on their correct discernment of their role in their particular community. USENIX's success and strength has been achieved by enabling and enhancing communication among all levels of UNIX users: through conferences, *;login:*, Usenet mapping, Stargate, manual reproduction, tutorials, and workshops. USENIX must continue to engender projects like these and continue to monitor and improve their quality.

I believe USENIX should continue to foster the positive communications that are so visible to the community. I hope that moderators and editors can be found in order to maintain and improve the technical quality of the conferences, tutorials, and newsletters. New projects which contribute services or products to the UNIX community are candidates for future USENIX support.

I advocate the creation of more focused technical workshops (patterned after the Rhode Island networking workshop), refereeing of some of the papers for USENIX conferences (e.g., those of some particular focus for a part of a conference), and vigorous solicitation of high-quality articles for *;login:*. I believe I can help foster the communications that USENIX needs.

Kirk McKusick

CANDIDATE FOR DIRECTOR

Background Information

Kirk McKusick got his undergraduate degree in Electrical Engineering from Cornell University. His graduate work was done at the University of California, where he received Masters degrees in Computer Science and Business Administration, and a Ph.D. in the area of programming languages. While at Berkeley he implemented the 4.2BSD fast file system and was involved in implementing the Berkeley Pascal system. He currently is the Research Computer Scientist at the Berkeley Computer Systems Research Group, continuing the development of future versions of Berkeley UNIX. He is a member of the editorial board of *UNIX Review* and a member of ACM and IEEE.

Position Statement

The goal of USENIX Association as stated in its charter is the "exchange and communication of research and technological information and ideas pertaining to UNIX and UNIX-related computer systems." The strength of the organization has been its emphasis on presenting the leading edge of UNIX research and technology. USENIX should continue to have conferences with refereed papers and proceedings available at or before the conference. It should continue its technical services including the newsletter *;login:* and its software distributions. I am personally committed to revising and printing 4.3BSD versions of the popular USENIX manuals.

At the same time, the emergence of the commercial UNIX marketplace has necessitated large marketing-oriented conferences such as those run by /usr/group. Although many members of USENIX need or want to attend these conferences, USENIX should not try to host marketing-oriented conferences. An appropriate compromise is to work with /usr/group to schedule separate conferences with cross registration that are held at the same time and in the same city so that members can attend both conferences in a single trip.

Educational, research, and advanced commercial development communities have a major role in the development and promulgation of UNIX. As a researcher at an academic institution, my viewpoint will perform a key role in representing these interests.

Michael D. O'Dell

CANDIDATE FOR DIRECTOR

Background Information

B.S. in Computer Science, University of Oklahoma, 1976
M.S. in Computer Science, University of Oklahoma, 1980

1983-Present
Senior Computer Scientist and Director of R&D,
Group L Corporation, Herndon, Virginia.

Group L builds software and systems products for the Information Industry. The products run on UNIX systems as well as popular personal computers.

1980-1983
Computer Scientist, Computer Science and Mathematics,
Lawrence Berkeley Laboratory, Berkeley, California.

Conducted research, supported the UNIX research computing environment at LBL, and consulted with other Department of Energy National Laboratories installing UNIX systems. Supported a large community of off-site ARPAnet users managing DOE research programs via electronic mail and teleconferencing hosted on the LBL machines. Participated in various ARPAnet protocol working groups.

1977-1980
Coordinator, Engineering Computer Network, College of Engineering,
University of Oklahoma, Norman, Oklahoma.

Guided the acquisition and installation of the first independent computer system on the University campus. Provided UNIX timesharing support for a community of 1500-2000 students and faculty.

Position Statement

Just like UNIX itself, the USENIX Association has been maturing rapidly over the last few years. The transition from a largely-volunteer organization, which needed only a minimum of mechanisms and procedures, to a more structured organization commensurate with its large cash flow and diverse membership community has been occasionally painful, but necessary. This evolution will continue, and it is essential that the organization supporting the membership not lose its vitality. Guiding the process through this awkward time is one of the principal concerns of the Board.

I believe USENIX has a mandate to maintain its position as the rallying-point for the technical forces within the UNIX community. This is a serious responsibility which must be supported by

continuing the tradition of technical excellence at the large semi-annual meetings, but in addition, by providing new avenues for technical interchange. The very sucessful USENIX Graphics Workshops and USENIX's pioneering support of the Stargate Experiment are prime examples of this continuing commitment. Clearly these efforts can be built upon to expand the value of the Association to the membership.

Finally, I personally believe that doing new, innovative things is a fundamental piece of "the UNIX philosophy" (whatever that might be). And while the USENIX community is clearly much wider and encompasses more diverse concerns than the technical core which founded the organization, the Association as a whole must remain committed to doing new and innovative things while retaining a real sense of pragmatism. Gratuitous change is damaging; no change is death.

Thank you for this opportunity to share my views.

John S. Quarterman

CANDIDATE FOR DIRECTOR

Background Information

John S. Quarterman first used UNIX at Harvard, where he got a B.A. in 1977, and learned to like it at BBN when he worked there afterwards. He is currently chief programmer and manager of the UNIX programming staff at the Department of Computer Sciences at the University of Texas at Austin. He continues to pursue his interests in networking, particularly in the ARPA Internet TCP/IP protocols and their use in a campus-area network. He is the representative of USENIX to the IEEE P1003 UNIX Standards Committee and moderates mod.std.unix on Usenet. His publications include:

"Notable Computer Networks," John S. Quarterman, Josiah C. Hoskins, forthcoming in 1986.

"4.2BSD and 4.3BSD as Examples of the UNIX System," John S. Quarterman, Abraham Silberschatz, James L. Peterson, *ACM Computing Surveys*, December 1985.

"UNIX System V and 4.1C BSD," John B. Chambers and John S. Quarterman, *Proceedings of the Summer 1983 (Toronto) USENIX Conference.*

Position Statement

As a USENIX Board member from a university I hope to help preserve some of the original academic character of the Association. UNIX is big business now but there exist other organizations more specifically oriented towards the interests of vendors and of commercial end users. It is good that USENIX gives tutorials and publishes papers related to the business market. But the primary emphasis of USENIX Conferences should be on technical papers, and tutorials should be designed keeping in mind that research and academic users do not necessarily run the same systems as business

users. Good technical tutorials are the objective regardless of vendor size or market share of product. The larger the company and the greater their support for a product, the less need for a USENIX tutorial in addition.

USENIX is already involved in most appropriate ventures. However, there may also be more good publishing projects like the 4.2BSD manuals, and I would like to see it take a stronger role in organizing the UUCP and Usenet networks.

Michael Tilson

CANDIDATE FOR DIRECTOR

Background Information

B.Sc. Physics and Mathematics, University of Michigan, 1973
M.Sc. Computer Science, University of Toronto, 1975.

Over 10 years of UNIX experience, including installation of a UNIX Fifth Edition system. Attended the first meeting of what would eventually become USENIX, and has been active in UNIX user groups ever since. Program chairman for the Summer 1983 Toronto USENIX Conference. Member of the USENIX Board of Directors since 1984. Board member responsible for USENIX tutorial coordination. Currently president of Human Computing Resources Corporation.

Position Statement

In the coming years, the USENIX Association should continue to improve as a technical/professional group. In the past two years, progress has been made on several fronts:

- o USENIX has a much clearer direction. We are no longer confused with /usr/group or other trade associations. We now have a clear purpose as an organization of technical professionals.
- o The quality of our conferences and tutorials has improved substantially.
- o USENIX has sponsored various experiments to serve our community, including the net mapping project, Stargate, small technical workshops, etc. New services, such as the 4.2BSD manual service, have been instituted.
- o USENIX management has become more professional, and USENIX has been able to cope with rapid growth.
- o The USENIX financial position is strong, which allows the Association to take bigger risks in order to benefit the members.

In the next term of the Board, I believe the following are important issues:

- o The quality of our technical conferences and tutorials must be maintained and improved in the face of explosive growth in the number of UNIX users. We must aim for quality – for the next few years we will grow in numbers without any effort at all, and we must not pursue size for its own sake.
- o The quality and usefulness of *;login:* must be improved. Some improvements have been made, but we can do more.
- o Now that the Association has reasonable financial resources, we should look for new services that can be provided to the membership. We should continue to support new experiments and try new ideas. This must be done in a controlled way, and we must not forget our organizational goals, so we can't support every good idea that comes along. But I think the time has come to expand our scope a bit.
- o USENIX must continue to be well-managed. The Association has been remarkably free from the squabbles, disruption, and in-fighting that is so common in similar organizations. We have also been free of the financial crises that are often found in professional organizations. If we are to continue to serve the members, this stability must continue.

As a member of the USENIX Board of Directors in the last two years, I have been pleased to be a part of the growth of the Association. It has been especially gratifying to work on the tutorial program, which was my particular responsibility, and which has become quite successful. Working with the other Board members, the office, and the membership has been a pleasure more often than I had expected. If I am chosen to serve again, I would enjoy the opportunity to help the Association continue its progress.

David Yost

CANDIDATE FOR DIRECTOR

Background Information

I started out in computers in 1975 doing microprocessor hardware and software design. My initiation into the world of UNIX came in 1977 when, lacking access to a UNIX system, I read a set of Version 6 manuals cover to cover to see what UNIX was about. (How many new users start out that way nowadays!?) I then worked on the Rand Editor and did other UNIX work for the Rand Corporation, did UNIX kernel and utilities consulting, worked in the Fortune Systems operating system group, did more UNIX consulting, and now I'm back working on a still better version of the Rand Editor.

Position Statement

I am the "USENIX can be fun" candidate.

I have been attending USENIX conferences since 1979. At the conferences in 1983, I organized first an Omnimax movie marathon in San Diego, and then an Imax movie marathon in Toronto for conference attendees. At the June, 1985 Portland conference, I organized the fireworks for the conference picnic celebrating the tenth anniversary of USENIX. You may have seen me listed as "Rev. David Yost" in connection with my recent paper at the Portland conference in June, 1985. Ear Reverend would be more like it. Actually, this little joke was foisted on me by the Very Ear Rev. Dr. Greg Chesson, the session chairman for the conference, in reference to a paper I gave on the Rand Editor a few years ago. For that talk, I took to heart the religious character of the endless debate about favorite text editors and attempted, in the style of a TV preacher, to convert the Sinners of the Other Editors. In Portland, it came to pass that I Rev.'ed the tables on Dr. Greg and put him up to introducing the plenary speakers "from the piano" which he carried off marvelously.

Enough hystery about my past involvement in extracurricular fun at USENIX. Board members give a lot of effort to guiding the main activities of USENIX, so I would like to give you some of my thoughts on those.

I think that USENIX has evolved very gracefully while the position of UNIX in the world of computing has evolved, and has enjoyed a lively mix of good content and good cheer. I have been particularly happy to see some of the new trends, such as the professionally-published proceedings available before the start of the conference, the high-quality tutorials, and the relatively recent professional organization of the conference logistics.

I would like to see USENIX continue to be a place where users share technical information about interesting things they do on UNIX as well as to UNIX. Inevitably, some of the work done to UNIX has led and will lead away from the standard versions. I hope we can continue to accomodate new trends and be a home for people who want to share new work that grows out of UNIX.

The present board is very motivated toward searching for new ways to serve USENIX members, and I look forward to participating in carrying on this thrust. The last thing I want to see is for USENIX to go stale.

If I am elected to the Board of Directors, I will work toward keeping USENIX alive and well — and fun.

GKS Code Being Distributed as First 1986 Software Tape

In April, the 86.1 software distribution tape will be sent to all 1986 Institutional and Supporting members who have sent in their 1986 tape release form.[1] Since this distribution does not contain any material covered under UNIX licenses, it will be sent to all qualified Institutional or Supporting members, whether UNIX-licensed or not, on the usual right-to-use basis.

The 86.1 tape contains a single contribution: Sandia National Laboratory's implementation of the Graphical Kernel System (GKS), written by Randall W. Simons. GKS is both an American National Standard (ANS) and an International Standard graphics package. This implementation includes the lowest level of the ANS (level ma) and some of the routines from level mb. It conforms to ANS X3.124-85, and to the May, 1985, draft of the C Binding of GKS being developed by ANSI X3H3. The package supports the following graphics capabilities: 2D lines, markers, text, and filled areas; control over color, line type, and character height and alignment; multiple simultaneous workstations; multiple transformations; and locator and choice input.

An implementation of the Small Plot Package (SPP) is also included. It provides single-call plot commands, linear and logarithmic axis commands, control over tick marks, tick mark labels, and plots data with or without markers and connected lines. Some of the level mb functions have also been implemented, and manual pages and code templates for all GKS functions (not just level ma) are included.

The package was written in C on a VAX 11-780 running ULTRIX[2] (4.2BSD). It should be easy to port to other systems where the C compiler supports the same features. The package supports Tektronix 4014 and 4115 terminals, but is designed to easily add support for other device types.

Announcing the Fourth Printing of USENIX 4.2BSD Manuals

Since inventories of the third USENIX-sponsored printing of 4.2BSD manuals are sold out, and since the office has continued to receive orders, USENIX has decided to make a fourth manual printing. Shipments are expected to begin after the printing has been completed — sometime after April 7, 1986.

The fourth printing will be identical to the previous ones. Prices remain unchanged, and procedures for ordering the manuals will remain as before. A 4.2BSD Manual Reproduction Authorization and Order Form may be found near the end of this newsletter. It is important to fill out this form carefully, include a check or valid purchase order, and have your USENIX representative sign the form. As before, only current Corporate, Institutional and Supporting members can order manuals. Failure to follow the procedures detailed on the order form can cause a delay in processing your order, so please be sure to include all the information asked for.

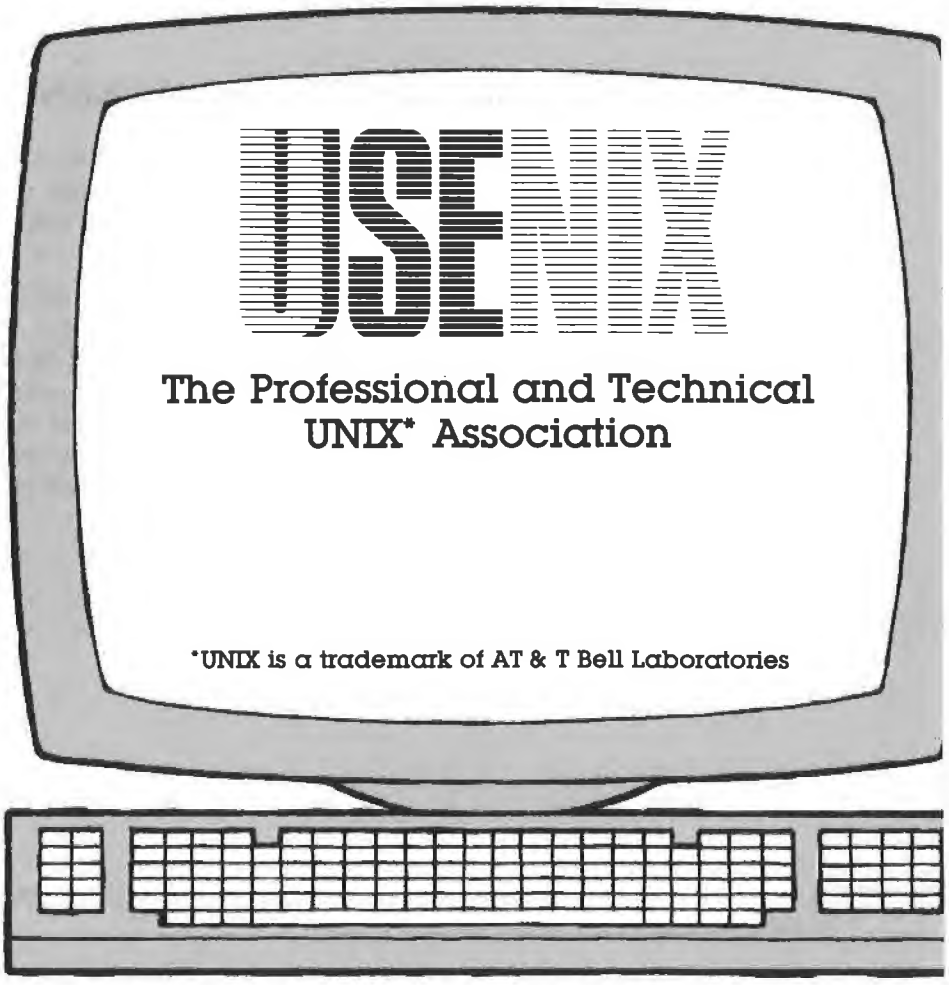
[1] A 1986 tape release form was mailed with the Institutional and Supporting member renewal forms.

[2] ULTRIX is a trademark of Digital Equipment Corp.

page for atlanta artwork

Summer 1986 USENIX Conference & Exhibitic

Atlanta



June 9-13, 1986
Atlanta, Georgia

The Summer 1986 USENIX Conference

The Summer 1986 USENIX Conference and Vendor Exhibition will be held on June 9-13 at the Atlanta Hilton and Towers in Atlanta, Georgia. A pre-registration package containing detailed Conference information and hotel reservation forms will be mailed in early April.

The conference host is Medical Systems Development Corporation. Paul Manno is the local coordinator.

Tutorials – June 9-10

A variety of tutorials will be offered; they are discussed in the next article in this issue of *;login:*. Please remember to register early, as all tutorials at the Winter Conference were sold out.

Technical Sessions – June 10-12

Technical presentations including, but not limited to, the following areas are scheduled:

- The Audio-Visual UNIX
- Computer Typography
- User Interface Technology
- Electronic Mail Systems
- Operating Systems Design
- Remote Filesystems
- The Philosophy and Theology of UNIX

The members of the Program Committee are:

Mike O'Dell, Group L Corporation, Chair
seismo!mo or mo@seismo.arpa
(703) 471-0030 or 378-8574
John Chambers, MCC
Mike Hawley, Lucasfilm Ltd.
Sam Leffler, Lucasfilm Ltd.
Jim McKie, Bell Communications Research
Dennis Ritchie, AT&T Bell Laboratories
Spencer Thomas, University of Utah
Computer Science Department

Vendor Exhibition – June 10-12

There will be a vendor exhibition with displays of advanced technology relevant to the UNIX technical community. If you wish to receive details on exhibiting at Atlanta, please contact:

John Donnelly
USENIX Exhibit Office
Oak Bay Bldg., 4750 Table Mesa Drive
Boulder, CO 80303
(303) 499-2600

For Further Information

If you did not receive this announcement directly and wish to be on the mailing list for receipt of the pre-registration packet, please contact:

USENIX Conference Office
P.O. Box 385
Sunset Beach, CA 90742
(213) 592-3243 or 592-1381

Tutorials to be Presented at the Summer '86 USENIX Conference

UNIX Technology from the Experts

The USENIX Association is once again offering its well-respected program of intensive UNIX tutorial sessions. These sessions are not "market overview" discussions – the tutorial sessions are taught by leading experts, are aimed at an audience of software professionals and technical managers, and should be immediately applicable to UNIX systems development and maintenance. This is your opportunity to learn from an expert at a reasonable cost and at a convenient time.

An expanded format will be used for the USENIX tutorial program in Atlanta. As a result of the tremendous demand at our January conference in Denver, Colorado, the most popular tutorials (Introduction to 4.2/4.3BSD Internals, Advanced Topics on 4.3BSD Internals, UNIX System V Internals, UNIX Device Drivers (4.2BSD), and Local Networks) will be presented on Monday, June 9, 1986 AND Tuesday, June 10, 1986. By offering these full day tutorials on both days we hope to allow more people to attend these continuously "sold out" tutorials.

The tutorial program has also expanded into new topic areas that we expect to be well-attended by the UNIX community. Our popular traditional tutorials will also be offered. Attendance will be limited, and pre-registration is strongly advised. On-site registration will be allowed ONLY if space permits. The pre-registration packet for the Atlanta conference will be mailed in early April.

INTRODUCTION TO 4BSD UNIX SYSTEMS ADMINISTRATION

Ed Gould
MT XINU

The basics of administering a 4BSD UNIX system will be covered. The tutorial will be oriented mainly towards Berkeley VAX UNIX systems. Topics covered will include system startup and shutdown, resource management, performance and tuning, the UNIX file system, and security, as well as others. The tutorial is designed for systems administrators, not for systems programmers. A rudimentary knowledge of UNIX is assumed.

Ed Gould has been working with UNIX since 1976. At the Computer Center at the University of California in Berkeley, he was involved with the management and administration of several systems that were used for general purpose timesharing for the campus. In 1983, along with Vance Vaughan and Bob Kridle, he founded mt Xinu, a company dedicated to the support and enhancement of technically advanced UNIX systems.

INTRODUCTION TO SYSTEM V UNIX SYSTEMS ADMINISTRATION

Rebecca Thomas and Rik Farrow
UNIX/WORLD

This tutorial will cover the fundamentals of administering a UNIX System V system. The material is intended for administrators, not systems programmers. Attendees should understand the UNIX system at the user's level and be able to read simple Bourne shell scripts. We will discuss the file system – its constituent parts, how to create new file systems, how to interpret *fsck* reports, managing disk space, and backing up and restoring files. We'll also discuss system startup and shutdown, controlling system access (includes disabling/enabling terminal lines, adding and removing accounts). Along the way, we will cover UNIX system security, the line printer spooling system, and UNIX communications (*cu* and *uucp*).

Dr. Rebecca Thomas is technical editor of *UNIX/WORLD* magazine and is author of two guide books on UNIX, *A User Guide to the UNIX System* and *Advanced Programmer's Guide to UNIX System V*. She is currently writing a book on UNIX system administration for Prentice-Hall.

Rik Farrow is a UNIX systems consultant and writer. He has written UNIX installation manuals for three different computers, and numerous magazine articles. He and Dr. Thomas are currently writing a guide book on UNIX System Administration for Prentice-Hall.

SOFTWARE DEVELOPMENT USING C AND UNIX

Rob Kolstad
CONVEX COMPUTER CORPORATION

This tutorial is for programmers who wish to use the full extent of software development tools available under UNIX. Programmers just getting into UNIX and C will be exposed to sets of tools which will allow them to exploit UNIX and – if they are used to another operating system – be able to do all those things they used to be able to do before. Proficient programmers might find their productivity increased as they exploit the available tools more effectively. One of the course's goals is to disseminate all those little "tricks" that experienced programmers know but which are written nowhere. The course is not a C tutorial, though it would make a fine conjunct for programmers who are learning C and need to learn the surrounding environment in order to make the smooth transition to the UNIX software engineering environment. The tutorial includes: comments on the UNIX philosophy, shell programming, writing and debugging C programs (including the preprocessor, *lint*, argument processing, debug schemes, conventions, symbolic debugging, and *adb*), optimization and profiling techniques, advanced problem-solving techniques (including plagiarism, configuration programs, file manipulation, and high level tools), project management techniques (including RCS), and hints on documentation. Extensive examples and explanations will accompany the text for this course.

Dr. Rob Kolstad has developed UNIX software for ten years – starting back in the days of Version 6. He currently manages both the operating system and computer operations groups at Convex Computer Corporation in Richardson, Texas. He received his Ph.D. in systems programming languages from the University of Illinois and is a frequent speaker at USENIX Conferences.

WRITING PORTABLE C PROGRAMS

Tom Plum
PLUM HALL, INC.

Today the C programming language is widely used to implement portable applications programs. But there are many pitfalls for the unwary, some obvious, but some very subtle. If you are not aware of the issues, it is easy to write programs that will not operate correctly in another hardware architecture, or another UNIX version, or another version of the C compiler. It then becomes expensive to move the application to a new machine. This course will teach you to recognize the trouble spots and avoid the pitfalls. You will learn to write truly machine and system independent code, and to protect yourself when this is not possible. This course is intended for experienced C application developers. If you are involved in the development of software which is to be used or distributed of a variety of systems, you should take this course.

Tom Plum is chairman of Plum Hall, Inc., a publishing and training firm specializing in the C language. He is the author of two textbooks on C. Dr. Plum is also vice-chair of the ANSI X3J11 C Language Standards Committee.

ADVANCED UNIX PROGRAMMING IN C

Carol Meier
EMERGING TECHNOLOGIES

This tutorial will explore the details of processes in UNIX. While most of the concepts will apply to both AT&T and Berkeley UNIX, the implementation will be specific to System V UNIX. It is intended for programmers interested in understanding the following features of the UNIX system interface: how to use system calls, attributes of UNIX processes, process creation, environment manipulation, setting up pipes, implementing I/O redirection and background processing. A working knowledge of C language basics (basic data types, operators, expressions, statements, functions, simple arrays) is assumed. Advanced C topics necessary to understand the tutorial will be covered at the beginning.

After receiving her B.S. and M.S. degrees in Computer Science from the University of Pittsburgh, Carol Meier worked for Bell Laboratories as an applications programmer. Since 1983 she has worked as an independent consultant specializing in UNIX and C training and software development. She has presented dozens of public and on-site professional technical seminars throughout the United States and Canada. She has developed and currently teaches hands-on UNIX and C courses at the University of Colorado.

LANGUAGE CONSTRUCTION TOOLS ON THE UNIX SYSTEM

Stephen C. Johnson
THE DANA GROUP

This tutorial is intended for C programmers who want to become familiar with the language development tools available on the UNIX system. The course will be directed towards application designers who may wish to use these tools to make front ends for their applications, rather than towards "traditional" compiler writing. Specific topics covered include: designing a language recognizer, the *lex* and *yacc* programs, symbol table issues, error reporting and recovery, strong type checking, and testing. Several in-class exercises will be given to lead the students through the construction of a simple front end.

Steve Johnson received his Ph.D. degree in pure mathematics from Columbia University in 1968. In 1967, he joined Bell Laboratories, Murray Hill, N.J., where he worked in psychometrics, computer music, and the computation center before joining the Computer Science Research Department. As a researcher, he worked on computer algebra, wrote the *yacc* parser generator, contributed to complexity theory and the theory of code generation and parsing, wrote the Portable C Compiler, and for several years, was involved in experimental VLSI design and silicon compilation. In 1983, he accepted a position with the AT&T computer line of business and was head of the Language Development Department at AT&T Information Systems. In 1986, he was named a Bell Laboratories Fellow. Dr. Johnson is currently Director of Programming Languages for The Dana Group in Sunnyvale, California.

INTRODUCTION TO 4.2/3BSD INTERNALS

Thomas W. Doeppner, Jr.
BROWN UNIVERSITY

4.2BSD SOURCE LICENSE REQUIRED FOR THIS TUTORIAL

This tutorial is an introduction to 4.2BSD and 4.3BSD internals. It is geared to the programmer with a good knowledge of UNIX programming in C, but with little or no experience with UNIX internals. The course will cover process management, high-level I/O (including the file system), low-level I/O (i.e., device drivers), virtual memory, interprocess communication and networking. After taking the tutorial, the individual will have a basic knowledge of the structure of 4.2/3BSD and should be able make his or her way through kernel code.

Thomas W. Doeppner Jr. received his Ph.D. in Computer Science from Princeton University in 1977 and has been on the faculty at Brown University since 1976. He has lectured extensively on UNIX internals over the past two years for the Institute for Advanced Professional Studies.

ADVANCED TOPICS ON 4.3BSD INTERNALS

Marshall Kirk McKusick and Mike Karels
UNIVERSITY OF CALIFORNIA, BERKELEY

4.2BSD SOURCE LICENSE REQUIRED FOR THIS TUTORIAL

This tutorial is directed to systems programmers who have taken a course on 4.2BSD internals or who have had at least a year of experience working on the 4.2BSD kernel. The tutorial will cover the performance work done for 4.3BSD and will also discuss recent and planned changes to the kernel interfaces and facilities. The intent of the tutorial is to present a wide variety of material at a descriptive level. Presentations will emphasize code organization, data structures, and algorithms.

Kirk McKusick got his undergraduate degree in Electrical Engineering from Cornell University. His graduate work was done at the University of California, where he received Masters Degrees in Computer Science and Business Administration, and a Ph.D. in the area of programming languages. While at Berkeley he implemented the 4.2BSD fast file system and was involved in implementing the Berkeley Pascal system. He currently is the Research Computer Scientist at the Berkeley Computer Systems Research Group, continuing the development of future versions of Berkeley UNIX.

Mike Karels received his B.S. in Microbiology at the University of Notre Dame. While a graduate student at the University of California, he was the major contributor to the 2.9BSD release of the Berkeley Software Distribution for PDP-11s. His current position is Principal Programmer at the Berkeley Computer Systems Research Group, continuing the development of future versions of Berkeley UNIX.

UNIX DEVICE DRIVER DESIGN (4.2BSD)

Daniel Klein
CONSULTANT

4.2BSD SOURCE LICENSE REQUIRED FOR THIS TUTORIAL

This course is designed for people who wish to become familiar with the fundamentals of designing UNIX device drivers. A knowledge of the major structures and internals of 4.2BSD UNIX is a desirable prerequisite for this tutorial, although a specific knowledge of the finer details is not required. This tutorial will cover the major aspects of driver design and implementation, and device integration. Both DMA and programmed I/O device drivers will be covered, as well as block and character (buffered and unbuffered) interfaces. We will outline the design and implementation of structured I/O devices (i.e., disk drives), and semi-structured devices (i.e., tape drives and serial communication links). This course will also discuss all aspects of adding a new device to the kernel (i.e., autoconfiguration, special files, device tables, and debugging). The intended audience for this course is systems programmers who will be actively engaged in the maintenance or design and implementation of UNIX device drivers. Although this course will be geared towards 4.2BSD, a comparison between the Berkeley and Bell Labs approaches will be offered. Users of System III or System V will therefore find this course to be informative.

Daniel Klein has been involved with UNIX since the original university distribution of Version 6 in 1976, including writing device drivers, utility programs, applications systems, and enhancements to the kernel. A graduate of Carnegie-Mellon University, Mr. Klein was manager of software systems at Mellon Institute for six years. He is presently engaged in teaching UNIX internals and developing an on-line educational system for UNIX, as well as developing a multi-processor simulation system.

UNIX SYSTEM V INTERNALS

Maury Bach and Steve Buroff
AT&T INFORMATION SYSTEMS

SYSTEM V SOURCE LICENSE REQUIRED FOR THIS TUTORIAL

This tutorial is a survey of the internal structure of AT&T's UNIX System V, and it is intended for people who maintain, modify or port UNIX systems. The tutorial will discuss traditional kernel concepts such as the file system, I/O subsystem, and process management, as well as new features in System V Release 3, such as demand paging, the file system switch, streams, remote file sharing and shared libraries. Attendees should have a good working knowledge of the UNIX system; basic kernel knowledge is recommended.

Maury Bach and Steve Buroff are Distinguished Members of the Technical Staff at AT&T Information Systems. Maury Bach has worked on multi-processor UNIX system development and streams. Steve Buroff has worked on multi-processor development and on paging virtual memory implementations. Bach has also taught a multi-week UNIX internals course within Bell Labs and is the author of a forthcoming book, *The Design of the UNIX Operating System.*

AIX ON THE RT PC

Charlie Sauer and Larry Loucks
IBM

This tutorial gives the software developer an overview and background material designed to allow him or her to effectively develop software for the AIX operating system on the RT PC. Since the RT has a new hardware architecture, that architecture is first summarized. This includes a discussion of the RISC machine, 40-bit virtual memory, I/O system, floating point support, graphics support, and co-processing. The bulk of the tutorial is devoted to the software structure of AIX. This includes the relationship of AIX to standard UNIX System V, how AIX interfaces to the RT PC architecture, the Virtual Resource Manager, unique system services and their application, and the interfacing of device drivers to both AIX and the VRM. This tutorial presents a rare opportunity to learn about the structure of an important system as presented by senior architects of the system.

Dr. Sauer received his Ph.D. in computer science from the University of Texas at Austin in 1975. He is currently Manager of System Architecture for the IBM RT PC. Dr. Sauer has published three textbooks: *Computer System Performance Modeling,* co-authored by K. M. Chandy; *Simulation of Computer Communication Systems,* co-authored by E. A. MacNair; and *Elements of Practical Performance Modeling,* co-authored by E. A. MacNair.

Larry Loucks is a member of the IBM Senior Technical Staff and is the lead architect of the RT PC system. He received a BA in Mathematics from Minot State University in Minot, North Dakota. In his career at IBM since 1967, he has worked on a variety of products, including QTAM, TCAM, SNA, the 5520, and the RT PC.

LOCAL NETWORKS

Bruce Borden
THE DANA GROUP

This tutorial presents an overview of local networking technology, with emphasis on UNIX implementations and futures. It will cover the ISO Open Systems Interconnection Model, the newly emerging ISO Protocol Standards, IP/TCP, XNS (SPP), NETBIOS, X.25, and other "standard" protocols. Various physical layers will be covered, including IEEE 802.X standards, Ethernet, PC-net, Pronet, Hyperchannel, etc. Emerging distributed file system implementations will be reviewed with an emphasis on their changing demands on protocol and media requirements. Finally, network performance will be addressed.

Bruce Borden is Vice-President of Graphics for The Dana Group, developing a personal supercomputer. Prior to The Dana Group, Mr. Borden was Director of Engineering for Silicon Graphics, founder of 3Com, developed the Excelan TCP/IP front-end protocol package, and authored the Rand MH mail handling system.

MANAGING A LOCAL AREA NETWORK

Evi Nemeth and Andy Rudoff
UNIVERSITY OF COLORADO, BOULDER

This tutorial is a summary of all the things we (and many others) have learned over the past couple of years in managing a growing local area network. It is intended for system administrators and others involved in planning, configuring, installing, and maintaining a networked UNIX facility. The course emphasizes 4.2/4.3BSD networks, yet includes issues that are global to all networks. Topics to be covered include:

network overview (5%);
buying and installing network hardware (15%);
BSD network software installation (15%);
non-BSD software/hardware installation (5%);
experience with various vendors' products – discussion (5%);
management issues, control, source code, users, resources, accounting (15%);
tools to make these chores easier (10%);
debugging – hardware and software (10%);
writing programs that use the network (15%);
mail (sendmail) and the network (5%);
TCP/IP protocol demo (1%).

Evi Nemeth is on the computer science faculty at the University of Colorado and has built the University's Engineering Research Computing Facility from a single VAX 11/780 to its present complement of over 50 machines.

Andy Rudoff is a computer science student who has been involved with the systems work concerning this network's growth for the past 4 years. He is also working at ViaNetix, Inc., developing network systems software.

THE NETWORK FILE SYSTEM (NFS)

Mark Stein
SUN MICROSYSTEMS, INC.

The Network File System (NFS) is a distributed file sharing service, designed for use in heterogeneous computer networks, which allows a client user to access files transparently across machine boundaries without regard to machine type or operating system used. Remote Procedure Call (RPC) and External Data Representation (XDR) are support protocols used by NFS and are available for use by other network services. With the growing support of NFS shown at UniForum 1986 in February (fifteen companies demonstrating inter-operability on UNIX System V and 4.2BSD, VMS, and MS-DOS), NFS is becoming an increasingly important force in UNIX networking. All NFS protocol specifications, together with RPC/XDR source code, are in the public domain. This tutorial is aimed at software professionals who would like to learn more about the technical details of NFS and related areas (RPC, XDR, and mount protocols, yellow pages data lookup service, transport and user interfaces, auxiliary services, and NFS administration). It will cover architectural, protocol, and some implementation details of the various NFS components. In addition, we'll look at NFS's future directions.

Mark Stein is a Project Leader in the NFS Consulting group at Sun Microsystems. He is in touch with all aspects of NFS in the course of his work with the NFS vendor community, including conducting workshops and doing technical consulting. Mr. Stein taught the "UUCP, Mail, and News" tutorial at several previous USENIX conferences.

WINDOWING SYSTEM IMPLEMENTATIONS FOR BERKELEY UNIX

David Rosenthal
SUN MICROSYSTEMS, INC.

The course covers window systems for Berkeley UNIX, examining four examples (Sun, Whitechapel, Andrew, and X) in considerable detail from the point of view of an application developer wishing to use one (or more). It discusses the techniques used to implement the systems, and their effects on application structure and performance, techniques for writing programs that port between systems, and the components that are missing from current systems. It also discusses the problems of porting window systems between displays and CPU's, and the problems of porting these systems to the System V environment.

David Rosenthal has been researching interactive graphics and user interfaces since 1968. He co-chaired the technical review of GKS, and was Associate Director of the Information Technology Center at Carnegie-Mellon University. He was one of the developers of the ITC's Andrew portable window system.

Future Meetings

EUUG Conference & Exhibition – April 21-24, 1986, Florence, Italy

The next EUUG Conference and Exhibition will be held in Florence, Italy, on April 21-24. It will have technical and industrial conferences, tutorials, and an exhibition.

The conference will begin with a plenary session on April 21. During this session there will be a debate on "Research and Technology Policy of Europe."

The technical conference will run for three days, from April 22-24. The main theme will be real applications of the UNIX system, and especially distributed filesystems. Notwithstanding this, papers on all subjects relating to the UNIX system will be presented.

An industrial conference will run from April 22-24, in parallel with the technical conference. The program is oriented towards people interested in understanding the trends of UNIX System V, and especially the application areas where UNIX is used or is suitable for use. The main objective is the discussion of real cases, important projects, and problems involved in different situations and environments. The first day will be devoted to international themes. The second day will be devoted to items of specific concern to the Italian market. The third day will be a "free session," without pre-screening.

Tutorials will be presented on April 21 and 22. Those on the first day will address advanced features of UNIX. The second day will be devoted to introductory material for local Italian needs.

The working group of the IEEE P1003 Committee will hold a meeting in Florence on April 16-18, just prior to the EUUG Conference to allow and encourage European UNIX users to give their input into the committee's review and recommendations.

The EUUG has decided to run two conferences per year, one of which will host the major European UNIX exhibition. At the Florence conference, there will be a three-day exhibition opening on April 22.

If you would like to receive booking information please immediately contact:

Mrs. Helen Gibbons — +44 763 73039
EUUG Secretariat — mcvax!ukc!inset!euug
Owles Hall
Buntingford, Herts. SG9 9PL, England

The Programme Chair is:

Nigel Martin — +44 1 482 2525
The Instruction Set — mcvax!ukc!inset!nigel
152-156 Kentish Town Road
London, NW1 9QB, England

USENIX 1986 Summer Conference and Exhibition – June 9-13, 1986, Atlanta, Georgia

The USENIX 1986 Summer Conference and Exhibition will be held in Atlanta on June 9-13, 1986. There will be a conference, tutorials, and vendor exhibits. Please see the information at the beginning of this issue of *;login:*.

USENIX 1987 Winter Conference and UniForum 1987 – January 20-23, 1987, Washington, DC

The USENIX 1987 Winter Conference will held at the Shoreham Hotel in Washington, DC, on January 20-23, 1987. It will be concurrent with UniForum 1987, which will be at the Washington Convention Center.

On Text

Rob Kolstad
The Humor Editor

Remember learning how to write? Not literature, penmanship. Think back to first grade. In order to teach printing, they gave you those giant pencils which your tight little fist could just barely grasp. They supplemented the #1 lead in the pencil with that paper with the funky blue lines. My teacher always reminded us not to eat the paper; "It's made from old rags."

The big challenge in those days was getting the letters on the paper. "Little circle; big stem": the formula for p's, b's, d's, and sometimes q's. "Make those circles come up to the dotted line." I was lucky just to make a circle, though the pencil's line was plenty big enough to hit dotted lines and maybe even part of my desk. Mistakes were corrected using giant erasers that ripped holes in the paper. This was not what I now consider to be a high-tech solution.

Third grade brought with it a new technical innovation: cursive writing. Connect all those letters and you can write much more quickly. Legibility, of course, took a back seat to speed. I sure could get that stuff on paper in a hurry. Unfortunately, not even I could read it. Pink erasers still expunged those errors; sometimes the pencils even had working erasers on them. Unfortunately, the erasers often became black and hard, thus reducing their utility.

I don't recáll ever enjoying writing. I remember getting back from summer vacation and entering the fifth grade. We were supposed to write the prototypical "What I did..." essay; I had forgotten how to write. No loss, as it turned out.

The big deal for sixth graders in Norman, Oklahoma, was ink pens. I was never privileged enough to use real fountain pens; I had enough trouble with regular old 19-cent BIC's. My main problem was getting the ink on the paper rather than on my hands. I wish someone would have told me that it is much easier to make pretty letters with ink if you put a soft pad under your paper instead of a hard wooden desk. High class essays had to be copied letter perfect for final submission; error correction was via "a single straight line passing through the word." Low tech, really.

No improvements in the written expressiveness scene for me until 10th grade. By then I'd become proficient at keypunches and line printers but it was only a hobby. Third hour of every day was like being in heaven: TYPING. Typing machines (a.k.a. "typewriters") have been around for almost 80 years now. In the past they were manual devices which had a large number of small round surfaces ("keys") located on the end of metal manipulators. Pushing a key resulted in a letter being imprinted on the paper. The letters came out in a row, spaced linearly, 12 per inch. Typing was not only faster than cursive (with ink, to boot), it was more legible! Before long I could type 100 words per minute (five times faster than one can hand-write).

Error correction, on the other hand, was painful. At 100 WPM, it's pretty easy to make, say, 50 MPM (mistakes per minute). Using correction type or "White Out" or a typewriter eraser is really slow when you're used to hitting 8 keys every second. The solution, of course, is to slow down and make no errors. I still find that difficult.

By the time I got to graduate school, text editors were commonplace. Text editors were just like typewriters except they were electronic. They had a backspace key! You just backspace and type over the error. If one had a lower case line printer, one could generate letter-perfect copy. A text formatter allowed justified letter-perfect copy. What could be better? It was now possible to generate lots of personalized memos, letters, or pieces of correspondence; generate long papers which included pieces of other papers; and in general make very nice composition in the grand style of typewriters. This was the peak of productivity for text processing.

Enter UNIX, *nroff, troff,* and fast computers. Good old *nroff* not only allowed us to do justification, it also had Greek characters, underlining, and fancy indentation, just to name a few features. With the –e option, it spread words with hundredth of an inch accuracy. No longer could one put characters on a page into a 56 by 80 matrix of characters. Now there were half lines, hundredth of an inch spacing, and processors for tables and equations. People were still productive,

but everyone wants to have perfect equations and tables. Measures of productivity nosedived. It's a good thing that C/A/T 432's were so expensive.

Enter laser printers. Now everyone wants to have typeset output. It looks just like it came from a book – so it MUST be good. Unfortunately, where before the matrix of characters had only grown, now the number of variables increased over tenfold. Different fonts, different slants, and different boldness combined with 1/300th of an inch accuracy to allow one to put a character 90,000 different places in every square inch of paper! The correct mix of italics, Roman, and bold was now the goal. CPU time, formerly in abundance, was consumed completely with tech writers, engineers, and secretaries each trying to make their output look "just as good as Ralph's." Where before a single pass through an editor and simple processor like *nroff* sufficed, now it takes a dozen runs just to make sure every 'i' is dotted perfectly and every line aligns correctly with its neighbors and corners.

This is all harmless, you might think. I assert otherwise. It wasn't good enough for very smart guys to be able to place all manner of sizes and styles of characters in a virtually infinite number of places. Some people felt they needed to design better characters! Donald Knuth set the study of Computer Science back over half a decade by taking time out to design MetaFont, a program which allows anyone with enough hardware to create their own character set. Now if you can't find just the shape of 's' that you like, you can spend six months and design one!

Do you think any of this moves information around more quickly? Have we improved since first grade? cursive? ink? typewriting? We probably have. But at what cost? What will we do next? How can we get back Don's lost years? I'm not sure.

They say color laser printers (300 DPI/3 ppm) will be available late this year. Oh boy!!!

A Report on the Accuracy of Some Floating Point Math Functions on Selected Computers

Technical Report GIT-ICS 85/06[†]

Eugene H. Spafford
John C. Flaspohler

School of Information and Computer Science
Georgia Institute of Technology
Atlanta, Georgia 30332-0280
(404) 894-3152

Software Engineering Research Center
Georgia Institute of Technology
Atlanta, Georgia 30332
(404) 894-3180

ABSTRACT

The UNIX operating system and the C programming language have gained a large following in recent years, especially in research and academic settings. C and UNIX-like environments are available on a wide variety of machines, from personal computers to mainframe computers; however, few, if any, of these implementations provide accurate floating point libraries, although users tend to believe they do. This paper presents the results of running a set of accuracy tests on more than a dozen different computer systems under various versions of UNIX and UNIX-like environments.

December 17, 1985

Revised January 10, 1986

1. Introduction

The UNIX[1] operating system has, in recent years, become one of the most popular and influential general-purpose operating systems in existence. UNIX and UNIX-like operating systems are currently available or planned on machines ranging from IBM PCs[2] to the Cray 2.[3] UNIX is especially popular in academic and research environments, partially due to its flexibility and ease of configuration, and partly due to the number of different hardware configurations on which it is available. It also seems to be gaining popularity in a number of application environments, ranging from small businesses to large engineering firms.

Many, if not all, of these UNIX implementations are done in the C programming language. C is not an easy language to master, nor is it a particularly "safe" language, in the sense that there is no strong type checking, the syntax and operator precedences are often confusing to any but expert users, and the language supports pointer types with almost no restrictions. These same features (which

[†]Funding for the original printing of this report was obtained from the School of Information and Computer Science, and from the Software Engineering Research Center.

[1] UNIX is a trademark of AT&T Laboratories.

[2] IBM PC is a trademark of International Business Machines.

[3] Cray 2 is a trademark of Cray Research, Inc.

cause Pascal and Ada[4] proponents to blanch in horror) are usually the very reasons that C is so well loved by large numbers of UNIX programmers — the lack of restrictions and range of options provide the programmer with more power and possibilities than "safer" languages.[5]

The usual implementation of the C language under UNIX includes a library of standard mathematical functions for calculating various quantities like the sine and the square root of real numbers. The original language specification[6] does not specify the routines expected to be available in the runtime environment of the language. It does specify that all real number calculations be performed in double precision mode, but that specification does not identify the accuracy and precision[7] of any particular implementation of the math library functions, nor does it specify how errors should be signalled or handled in such routines. Since C is usually associated with UNIX, the default routines and error handling are generally assumed to be what UNIX normally provides. Most recent reference and tutorial books for C and UNIX tend to also blur this distinction, if they bother to mention anything about the math library at all (*e.g.,* [Bour83], [Past85], [Harb84], [Geha85]).

It is our understanding that most of the current UNIX implementations of the math library functions are derived from earlier versions of the library. As new versions of UNIX have been released and ported to new systems, it is probable that these same routines have been recoded from sources based on those early versions, but never tested or improved. It is likely that some systems contain assembly language versions of these original routines "optimized" for speed. Unfortunately, such "optimizations" often result in loss of precision; not many programmers realize that $x+yx$ is **not** always the same as $x(1+y)$ when implemented in code.

Late in 1984, a curious incident was brought to the attention of one of us: a set of programs written in C to optimize systems of non-linear equations would fail to converge to an answer when run under the 4.2 BSD UNIX[8] aspect of OSx on a Pyramid 90x,[9] yet the same programs converged to (correct) solutions in a matter of seconds when run on a Vax 11/750[10] also under 4.2 BSD UNIX.[11] That anomaly intrigued us, since previous research [Spaf83] had identified at least one case where a set of supposedly accurate computer math routines was seriously flawed in certain instances.

After some initial experiments, we decided to perform a more extensive set of comparison tests on routines commonly available in the standard C libraries of various systems running UNIX. We soon expanded our scope to include other machines available to us running UNIX-like environments. These results are detailed below.

The authors would like to stress that these tests are by no means exhaustive, and neither of us believes we are qualified to make *definitive* claims about the absolute accuracy of the firmware and software we evaluated. We do, however, believe that the results of our testing reveal some definite causes for concern, especially among the many users of UNIX-like systems who have accepted the accuracy of their standard operations without question or testing. On principle, we would urge *everyone* requiring accuracy in their calculations to perform their own testing or demand well-documented test results from their machine vendors.

---

[4] Ada is a registered trademark of the U.S. Government (Ada Joint Program Office).

[5] Some detractors of C point to examples such as *The C Puzzle Book* [Feue82], as a (damning) comment on the language. C adherents note that no one has yet written anything in Pascal or Ada which (they believe) has the complexity or usability of UNIX or some of its utilities. The reader may find some of the discussion in [Feue84] to be a rather interesting introduction to this debate.

[6] *The C Programming Language,* [Kern78]

[7] We use the word *precision* to specify the number of bits or digits in a result, while the word *accuracy* is intended to be a measure of how many of the bits of precision are correct when compared to the correct result.

[8] BSD stands for Berkeley Software Distribution and refers to their releases of UNIX. The version of UNIX developed at the University of California at Berkeley is very widely used on machines in academic and research environments. It is beyond the scope of this report to discuss the differences and origins of the various versions of UNIX.

[9] OSx and 90x are trademarks of Pyramid Technology Corporation.

[10] Vax is a trademark of the Digital Equipment Corporation.

[11] Private communication to Eugene Spafford from David Pitts, Georgia Tech.

2. Methods

2.1. The Tests

The tests performed were derived from the tests provided with *Software Manual for the Elementary Functions,* [Cody80]. This book presents a number of random accuracy tests for various standard mathematical functions in a clear and readable manner. The explanations accompanying the tests are also helpful when evaluating special cases (of which we found many). The tests in the book are written in Fortran and intended for testing Fortran functions against the ANSI standard for Fortran. We are very familiar with both Fortran and C, so we converted the test programs to C for use in our testing rather than design and code new tests. We felt that it was of importance that we use tests which were readily available in a form easily understood by others wishing to perform their own tests but who might not have a sophisticated mathematical background. The test algorithms in the Cody and Waite book rely on simple mathematical identities which are easily proven and which are implemented in a straightforward way. They should be easy for novices to understand and use, as well as of sufficient depth and rigor so as to provide a set of extensive tests.

We were also concerned that any tests we might employ be known to be accurate so that we would not introduce any errors in coding or test design. Our previous experience with the Cody and Waite tests, and a survey of some available alternatives resulted in our selection of the tests used. One related point in favor of the Cody and Waite tests is that the book contains all of the necessary constants and algorithms which a programmer might need to implement his/her own library of functions. We performed just such an implementation to verify some of our tests.

The tests were taken from on-line versions in Fortran which had previously been used by one of us [Spaf83] and which had been tested under various Fortran dialects. The tests were then converted to C using a set of conversion programs and shell scripts written by the authors. Final editing was done by hand, and the resulting tests were then run on a number of machines to attempt to identify any errors introduced by the conversion process. The use of statements printing intermediate values and the verification of indicated errors against results produced by use of other methods (*e.g.,* calculators) convinced us of the accuracy of the conversions.

The random number generator used in the tests was based on the 4.2 BSD UNIX *random* library function. We modified the BSD version of the function to be more portable to other systems, and to produce results we could easily replicate in trials on those other systems. After making these modifications, we ran a number of tests, such as sequence, collision, and frequency tests [Knut81] using the algorithm on a number of the machines available so as to verify the reliability and "randomness" of the function.

All of the function tests were compiled using full debugging whenever possible. This was done to defeat any attempts at register optimization which a "smart" compiler might provide. Additionally, extraneous assignment statements and block delimiters (such as parentheses) were added to the code in critical locations to defeat any compiler register tracking.

Each test is designed to run a short subroutine which determines machine characteristics, such as exponent size and representation. The data from this subroutine is then used to calculate limits to be used in testing the various functions over selected argument domains. Each test iteration is performed over 5000 random arguments and the results compared with values calculated by other means within each program.[12] The tests then usually conclude with checks against values at extremes within the defined domain of the function and/or machine accuracy, and selected tests to test identification and reporting of errors. We included a number of intermediary checks in each test to verify correct calculation of limit values and comparison quantities.

[12] For instance, the result given by the UNIX function might be compared against a value obtained through iterations of a Taylor series. In general, the algorithms used in the verification are too "expensive" (in terms of execution speed) to be used in the coding of a production subroutine library, even though they may be very accurate. Since they are unlikely to be used in the actual library, they can be used as independent methods of calculating results for comparison.

2.2. The Machines

We ran the tests on fourteen different machines (with the Pyramid's dual operating system counted twice). The tests are represented in the tables below. Tests run under 4.2 BSD UNIX on a Vax 11/750, a Vax 11/780 and a Vax 11/785 yielded identical results; their values are shown in the tables under "Vax." We also ran the tests on an AT&T 3B2/300 and on an AT&T 3B20S.[13] The results were identical on these two machines and so these results have also been collapsed into one table entry under the label "ATT 3B."

We attempted to have the tests run on an IBM PC/AT running Xenix, and on a Perkin-Elmer 3210 running a port of System V, but difficulties in transferring our test software to those systems and getting the tests running prevented us from getting a complete set of results in either case.

In the following descriptions, we describe systems as having floating point hardware or a floating point accelerator. These terms have specific meaning. Generally, a system may have floating operations in one of *hardware*, *firmware*, or *software*. Software floating point is when all floating point operations are emulated in operating system or user application level code. Usually, attempts to use machine instructions (as might be output from a compiler) for floating point calculations results in some form of fault or trap to the software, which then calculates and returns a result just as if an actual machine instruction had performed the calculations. It is also sometimes the case that there is an option available to the compiler which will cause it to generate special code to handle the floating point operations inline rather than as machine faults. The Masscomp and AT&T 3B2 machines we tested behave in this manner.

A firmware implementation of floating point arithmetic implies that microcoded operations perform the floating point arithmetic. The Vax 11/750 is an example of a machine we tested which does this. Firmware implementations are usually quite a bit faster than software implementations.

A hardware implementation generally implies a special purpose set of circuits (or even an attached processor) whose sole purpose is to quickly calculate floating point arithmetic results. These implementations are often very fast and accurate. In our tests, the Pyramid and Sun both had this kind of floating point hardware. In this paper, when we refer to a floating point accelerator (or *FPA*) we mean a hardware implementation of floating point arithmetic.

In some of the machines tested, it was not obvious to us from reading the documentation whether a machine did its floating point calculations in hardware, firmware, or software. Where possible, we indicate the method used.

2.2.1. AT&T 3B

The tests were run on an AT&T 3B2 model 300 running System V, version 2.0, release 2 UNIX. The 3B20S tested was a model 2 running System V, release 2.0, version 2 UNIX and was equipped with an FPA. The tests were run on the 3B20S with the accelerator active and again with the accelerator deactivated – the results were exactly the same in both cases. The only observable differences in the results between the 3B20 and 3B2 were that the tests took nearly two days to complete on the 3B2, while taking under an hour on the 3B20Ss. The tests were appreciably faster on the 3B20 with the floating point accelerator active.

2.2.2. AT&T 7300

The tests were run on an AT&T UNIX personal computer running System V release 1 version 2 UNIX for the 7300. This machine probably does its floating point calculations in software or microcode.

[13] 3B is a trademark of AT&T Technologies.

2.2.3. CDC Cyber 180/855[14]

We ran the tests under VX/VE version 5.0 level 630 with the C/VE C compiler. This is supposed to be a UNIX System V implementation running under NOS/VE.[15] Although the Cyber has the largest potential word size at 128 bits (and thus the largest potential for extended precision), the VX/VE implementation only uses single 64 bit words to represent floating point values — single and double precision in C/VE are the same. As a result, due to the number of mantissa bits used in this mode of the Cyber, its representation had one of the smallest mantissas of all the machines tested. The Cyber does its floating point operations in hardware.

2.2.4. Fortune 32:16

The Fortune system on which we ran these tests was the Fortune model 32:16. This system was running For:Pro, a vendor-supplied version of UNIX which combined utilities from Version 4.1 BSD with a few utilities from System III UNIX, and was not equipped with a floating point accelerator board. The running time for the tests on this system was approximately twelve hours.

2.2.5. Masscomp

The Masscomp system on which we ran these tests was running an enhanced System III version of UNIX. It was not equipped with a floating point accelerator, and apparently performed all of its floating point operations in software.

2.2.6. Onyx

The Onyx[16] was a model C8002 running ONIX release 3.03 (a vendor port of UNIX System III with enhancements). The Onyx is a Zilog Z8002-based system manufactured by Onyx Systems, Inc. It was not equipped with a floating point accelerator and performed all its floating point operations in software.

2.2.7. Pyramid 90x

The Pyramid 90x is a RISC machine designed especially to run UNIX. The Pyramid operating system, OSx, appears as two co-resident versions of UNIX: BSD 4.2 and AT&T System V. The user selects one of these as the current environment, or *universe*, and all subsequent operations, including kernel calls, are performed according to that context. We ran the tests in both "universes" and (surprising to us) the results differed significantly. The results for each "universe" are listed separately in the tables that follow. The Pyramid on which we ran the tests had a floating point accelerator board and was running OSx 2.3.1a at the time of the tests.

2.2.8. Ridge

The tests were run on a Ridge 32C system running the Ridge Operating System (ROS) release 3.2. The Ridge is a RISC machine designed for scientific processing. ROS is described in the documentation along with mentions of UNIX, System V and BSD, but without any claims that it is actually a UNIX port. The documentation and command environment seem almost identical to a UNIX system on the surface, so we include this as a "UNIX-like" system.[17] The Ridge can be equipped with a floating accelerator board, but the one we tested was not so equipped at the time of the test.

---

[14] CDC is a registered trademark of the Control Data Corporation.

[15] Our Cyber 180/855 was running version 1.1.2a, level 634 at the time of the testing.

[16] Onyx is a trademark of Onyx Systems, Inc.

[17] If ROS is considered as a port of UNIX, then we must conclude that the version we used was buggy and inconsistent. We had to heavily modify some of the tests and support routines in order to get them to run on the Ridge. Some of the tests never ran to completion, and others went into infinite loops that required the machine to be restarted. It is not possible for us to say whether this was due to incompatibilities between UNIX and ROS encountered by our coding, or due to outright errors in ROS. These problems may very well have compounded some of the accuracy problems found in our testing.

2.2.9. Sun 2

The tests were run on a Sun 2/170 equipped with a Sky floating point accelerator board. The Sun was running release 1.2 of Sun UNIX, which is a version of 4.2 BSD UNIX with extensions. The Sun floating point system implements floating point representations as presented in the IEEE 754 draft standard for floating point arithmetic [Comm81].

2.2.10. Vax

The Vax 11/780 tested had a floating point accelerator installed; the 11/785 and 11/750 did all of their operations in microcode. All three machines were running release 3.0 of the Ballistic Research Laboratories (BRL) version of BSD 4.2 UNIX. There were absolutely no differences observable among the various runs, except that the 785 runs completed in less than half the time[18] of the 780 tests. The tests took almost the same amount of time to run on the 11/750 as they did on the 11/780.

2.3. Our Implementations

After coding and running some of the tests, we were alarmed that a few of them produced indications of large errors on every machine. We checked and rechecked our code, and then checked some intermediate results. All seemed to indicate that our test code was correct and that all of the machines tested actually **were** very inaccurate.

To convince ourselves that the tests were, in fact, correct, we used the information in the Cody and Waite book to implement a few of the functions ourselves. These implementations were all done in unoptimized C, with no particular care given to error recovery or speed of computation. Instead, we concentrated on coding the functions to produce highly accurate results within the defined domain of the functions. This included coding floating point constants into the functions as hexadecimal bit strings, and doing manipulations of the bits in the exponent instead of simple multiplications and divisions. We would expect anyone implementing a production library to take at least this much care, but to also include better error reporting and some code optimizations.[19]

The results of the tests when run with our function implementations are presented in the charts which follow under the heading "Ours." Our implementations were done on a Vax 11/750. The tests were also ported to a Vax 11/780 and run, and they produced results identical to those produced on the 11/750 (as we expected).

2.4. Basic Assumptions

To simplify matters somewhat, we made a few assumptions about the systems under test. Our first, very critical assumption is that all of the tested machines perform the four basic operations of addition, subtraction, multiplication and division of floating point numbers accurately and with no bit loss. It is often extremely difficult to detect such errors, and we did not wish to write and conduct such tests, even though we have previously encountered machines that do not meet this assumption [Spaf83].

Our second basic assumption is that the C compiler provided with each system was producing correct code for floating point operations. A frequent source of such errors, the optimizer, was purposely turned off in each case where we had that option.

It is possible that the errors we found on one or two machines might be attributable to one of these assumptions being wrong, although we do not believe that happened. If that is the case, however, there is even more reason to be concerned with the overall accuracy of operations performed on those machines. If a user suspects such a problem on his/her machine, s/he should

[18] It should be noted that when we refer to running time for the tests, we are referring to the clock time and not the computer (CPU) time used. Thus, the tests may run faster or slower on a system depending upon the number of other users on the system at the time; therefore, we do not recommend that you draw any specific conclusions about the speed of a system based on these results.

[19] This is one of those rare cases where programming in assembly language might be preferred over a high-level language. Register manipulations and bit operations can make these functions extremely fast and accurate if done correctly.

either perform their own, more exhaustive tests and/or get some form of reliable test documentation from the vendors involved.

3. The Results

The following tables represent the results of our tests. As previously explained, each main accuracy test was performed 5000 times with values chosen at random from the domain shown with each test.[20] The number of times that the function value was different from the calculated value is shown, as is the maximum error obtained during that trial. The *maximum bit loss* represents the calculated number of mantissa bits in error at the point of maximum error. Maximum bit losses of 2 or less generally show an accurate routine (obviously, a loss of zero would be excellent). The *root mean square* error is a form of "average" error measurement, used to normalize the magnitude of error obtained at different points within each test domain. The RMS bit loss represents an "average" number of mantissa bits lost over the given test domain. RMS losses of one or under are to be hoped for. A more complete discussion on these measures and the individual tests involved can be found in [Cody80].

Other comments and results are presented with each test.

3.1. The Arc Sine Function

The results obtained from running the tests on the arc sine and arc cosine (asin and acos) functions indicate that all of the tested implementations are very accurate. All of the tests returned zero differences and zero bits of loss. As such, we have not included the tables of results for this test.

This agreement with our test values could be the result of algorithm match (i.e., the algorithms used by the systems are the same as used by the test[21]). It could also suggest that all of the implementations have been done correctly; we would like to believe the latter.

In the special tests, the Sun, the Fortune, and the Pyramid/BSD tests of $\sin^{-1}(x)$ where x is very small (effectively, an identity test) all showed some small error while the tests of the other machines all showed no errors.

[20] The pseudo-random number generator was always started from the same state (same initial seed) on each machine. The generator used state tables of integers to calculate values, which were then divided into a constant to obtain the real-valued "random" value. This helped to ensure that, within machine precision, all tests were performed with the same sequence of real-valued "random" values.

[21] The tests all used Taylor series expansions with truncation beyond machine precision.

3.2. The Arc Tangent Function

The results produced when testing the arc tangent (atan) function evidenced very little RMS bit loss for all the machines. We do notice in these tests some minor losses in the Maximum Bit Loss category, but overall, the results are quite acceptable for all the machines.

Random Accuracy Test #1

Comparison of $\tan^{-1}(x)$ *vs.* $\sum_{i=0}^{i=inf} (-1)^i \times \frac{x^{2n+1}}{2n+1}$ *in the domain* (−0.0625,0.0625)

Machine	Times >	Times =	Times <	Maximum Error	Max. Bit Loss	Root Mean Square	RMS Bit Loss
ATT 3B	1059	2847	1094	3.4296e-16	0.62719	1.03495e-16	0
ATT 7300	937	3138	925	3.38403e-16	0.607895	9.40659e-17	0
Cyber (VX/VE)	6	1191	3803	1.41132e-14	0.990055	5.11977e-15	0
Fortune	1067	2838	1095	3.4296e-16	0.62719	1.03754e-16	0
Masscomp	1332	2640	1028	2.30062e-16	0.0511748	1.02719e-16	0
Onyx	1181	2738	1081	3.84936e-16	0.793768	1.09286e-16	0
Pyramid/ATT	4	867	4129	4.639240e-16	1.063039	1.863392e-16	0
Pyramid/BSD	5	1240	3755	4.420831e-16	0.993468	1.596807e-16	0
Ridge	1568	3432	0	2.21828e-16	0	8.02971e-17	0
Sun 2	767	3333	900	3.60339e-16	0.698505	8.90709e-17	0
Vax	816	3337	847	3.734e-17	0.437946	1.10336e-17	0

Random Accuracy Test #2

Comparison of $\tan^{-1}(x)$ *vs.* $\tan^{-1}\left[\frac{1}{16}\right]+\tan^{-1}\left[\frac{x-\frac{1}{16}}{1+\frac{x}{16}}\right]$ *in the domain* (0.0625,0.267949)

Machine	Times >	Times =	Times <	Maximum Error	Max. Bit Loss	Root Mean Square	RMS Bit Loss
ATT 3B	1423	2663	914	4.41486e-16	0.991518	1.12584e-16	0
ATT 7300	980	2480	1540	4.43832e-16	0.999164	1.21843e-16	0
Cyber (VX/VE)	1161	2893	946	1.43824e-14	1.01731	3.45384e-15	0
Fortune	1470	2668	862	4.43211e-16	0.997143	1.14691e-16	0
Masscomp	1310	2597	1093	4.24674e-16	0.935506	1.1301e-16	0
Onyx	1379	2588	1033	4.42611e-16	0.995191	1.19315e-16	0
Pyramid/ATT	2552	2021	427	4.436506e-16	0.998575	1.436459e-16	0
Pyramid/BSD	2672	2058	270	4.469174e-16	1.009159	1.426119e-16	0
Ridge	1741	2264	995	2.8264e-16	0.348117	1.26546e-16	0
Sun 2	971	2611	1418	4.33367e-16	0.964738	1.14935e-16	0
Vax	537	2872	1591	5.42064e-16	0.965684	1.33788e-17	0

Random Accuracy Test #3

Comparison of $2\tan^{-1}(x)$ *vs.* $\tan^{-1}\left[\frac{2x}{1-x^2}\right]$ *in the domain* (0.267949,0.414214)

Machine	Times >	Times =	Times <	Maximum Error	Max. Bit Loss	Root Mean Square	RMS Bit Loss
ATT 3B	2071	2371	558	6.23048e-16	1.48849	1.44402e-16	0
ATT 7300	200	1768	3032	6.33104e-16	1.51159	1.79171e-16	0
Cyber (VX/VE)	765	2351	1884	1.90924e-14	1.426	4.66991e-15	0
Fortune	2447	2139	414	6.23094e-16	1.4886	1.58754e-16	0
Masscomp	798	3345	857	4.13099e-16	0.895637	1.00503e-16	0
Onyx	1833	2382	785	6.00278e-16	1.43478	1.41382e-16	0
Pyramid/ATT	1811	2302	887	6.302780e-16	1.505139	1.446518e-16	0
Pyramid/BSD	4643	618	19	7.548711e-16	1.765381	2.661715e-16	0.261507
Ridge	3519	1455	26	4.22265e-16	0.927299	1.65734e-16	0
Sun 2	2285	2293	422	5.42824e-16	1.28964	1.47156e-16	0
Vax	2743	1968	289	6.547e-17	1.23806	2.03847e-17	0

Random Accuracy Test #4

Comparison of $2\tan^{-1}(x)$ *vs.* $\tan^{-1}\left[\frac{2x}{1-x^2}\right]$ *in the domain* (0.414214,1.00)

Machine	Times >	Times =	Times <	Maximum Error	Max. Bit Loss	Root Mean Square	RMS Bit Loss
ATT 3B	484	3641	875	4.43002e-16	0.996462	9.19133e-17	0
ATT 7300	1663	3198	139	5.56605e-16	1.32581	1.19481e-16	0
Cyber (VX/VE)	1147	2599	1254	1.55218e-14	1.12731	4.32752e-15	0
Fortune	434	3583	983	5.4591e-16	1.29781	9.75154e-17	0
Masscomp	457	4151	392	2.77342e-16	0.320819	6.43753e-17	0
Onyx	356	1378	3266	4.38251e-16	0.980907	1.55253e-16	0
Pyramid/ATT	713	3675	612	4.293067e-16	0.951159	8.876810e-17	0
Pyramid/BSD	66	784	4150	5.649680e-16	1.34732	2.729553e-16	0.297815
Ridge	873	3857	270	3.78279e-16	0.768601	7.95022e-17	0
Sun 2	665	3823	512	4.22399e-16	0.927757	8.27987e-17	0
Vax	630	3712	658	5.05246e-17	0.864208	1.09771e-17	0

In other tests, a test of the identity $\tan^{-1}(x)$ vs. x for x very small showed some small errors on the Fortune and the Pyramid/ATT, and zero differences in the rest. Calling this routine on the Ridge with a fairly large (but legal) quantity would hang the system and required us to reboot the machine to recover.

3.3. The Exponential Function

In testing this function on the systems, we observed some very poor results. The first test had good results overall; however, the Masscomp machine showed a loss of over one quarter of its precision in the RMS bit loss (over 14 out of 52 bits); its 20 maximum bit loss is also cause for alarm.

Random Accuracy Test #1

Comparison of $\exp^{(x-\frac{1}{16})}$ *vs.* $\frac{\exp^x}{\exp^{\frac{1}{16}}}$ *in the domain* (−0.284074,0.346574)

Machine	Times >	Times =	Times <	Maximum Error	Max. Bit Loss	Root Mean Square	RMS Bit Loss
ATT 3B	732	3465	803	3.37182e-16	0.0602678	8.91913e-17	0
ATT 7300	593	3496	911	3.12174e-16	0.491499	8.81255e-17	0
Cyber (VX/VE)	409	2038	2553	1.47041e-14	1.04923	4.76178e-15	0
Fortune	1603	1779	1618	5.62883e-16	1.34198	1.70995e-16	0
Masscomp	632	3744	624	3.66383e-10	20.6541	5.18144e-12	14.5102
Onyx	1620	1779	1601	5.84896e-16	1.39733	1.70211e-16	0
Pyramid/ATT	2215	2711	74	3.671678e-16	0.72559	1.141821e-16	0
Pyramid/BSD	2224	1881	895	6.799817e-16	1.614646	1.683727e-16	0
Ridge	472	3531	997	3.09655e-16	0.479812	8.74512e-17	0
Sun 2	1404	1926	1670	5.88766e-16	1.40684	1.62817e-16	0
Vax	1467	1968	1565	6.37056e-17	1.19864	1.9613e-17	0
Ours	955	3436	609	3.92501e-17	0.499919	1.11544e-17	0

In the second test, four of the machines showed significant bit loss in both categories. Interestingly enough, the Masscomp performed very well in this test, indicating that perhaps the Masscomp library function has an argument reduction error — the results of test number three also suggest this possibility.

Random Accuracy Test #2

Comparison of $\exp^{(x-\frac{45}{16})}$ *vs.* $\frac{\exp^x}{\exp^{\frac{45}{16}}}$ *in the domain* (−3.465736,−707.010124)

Machine	Times >	Times =	Times <	Maximum Error	Max. Bit Loss	Root Mean Square	RMS Bit Loss
ATT 3B	761	3351	888	6.63234e-16	1.57867	9.41534e-17	0
ATT 7300	777	3473	750	3.33622e-16	0.587363	8.93164e-17	0
Cyber (VX/VE)	344	1827	2829	2.07387e-14	1.54533	5.72021e-15	0
Fortune	3072	1	1927	-0.000e-308	0	0	0
Masscomp	638	3641	721	2.29454e-16	0.0473531	8.4163e-17	0
Onyx	3052	1	1947	6.28391e-14	8.14467	2.41966e-14	6.76781
Pyramid/ATT	2125	2681	194	3.964503e-16	0.83629	1.148343e-16	0
Pyramid/BSD	3123	3	1874	6.277667e-14	8.143235	2.414073e-14	6.764476
Ridge	747	3535	718	3.25605e-16	0.552272	8.6921e-17	0
Sun 2	2978	2	2020	6.28709e-14	8.1454	2.47088e-14	6.79803
Vax	2189	39	2772	8.71905e-16	4.97332	2.99803e-16	3.43317
Ours	698	3345	957	6.02952e-17	1.11927	1.22414e-17	0

The third test resulted in the most significant loss of precision. In fact, the Onyx and the Pyramid/ATT versions lost the full amount of their precision; this indicates that the result was (effectively) wrong in the first significant bit of the mantissa. The Masscomp implementation once again evidences problems, and the Fortune, Pyramid/BSD, Sun and Vaxen all show significant losses. Note that our implementation on the same Vax hardware suffers from negligible loss.

Random Accuracy Test #3

Comparison of $\exp^{(x-\frac{45}{16})}$ *vs.* $\frac{\exp^x}{\exp^{\frac{45}{16}}}$ *in the domain* (6.931472,709.677352)

Machine	Times >	Times =	Times <	Maximum Error	Max. Bit Loss	Root Mean Square	RMS Bit Loss
ATT 3B	753	3366	881	3.05578e-16	0.460693	9.2704e-17	0
ATT 7300	811	3422	767	3.4362e-16	0.629966	8.99298e-17	0
Cyber (VX/VE)	339	1822	2839	2.60315e-14	1.87327	5.69602e-15	0
Fortune	3066	1	1933	6.28613e-14	8.14518	2.50911e-14	6.82018
Masscomp	630	3615	755	3.64809e-10	20.6479	7.29512e-12	15.0038
Onyx	3097	1	1902	1.0	52	0.173859	49.476
Pyramid/ATT	2160	2609	231	1.0	52	0.031623	47.017108
Pyramid/BSD	3129	0	1871	9.552339e-14	8.748861	2.500744e-14	6.815364
Ridge	756	3499	745	3.07878e-16	0.471511	8.80405e-17	0
Sun 2	2999	0	2001	6.28239e-14	8.14432	2.47855e-14	6.8025
Vax	2680	16	2304	8.85325e-16	4.99536	4.48566e-16	4.01447
Ours	703	3313	984	6.05936e-17	1.12639	1.21891e-17	0

In other tests, all of the machines evidenced some error when comparing the values $\exp(x)$ against $\exp(-x)$ for values between one-half and two. None of the errors was particularly large, but as a whole indicate that there is some small error present. Our implementation showed 1 bit of loss on two out of five of these tests — far less than the others, and within acceptable error.

One of the special tests for argument reduction errors showed a severe error for the Ridge. We do not know whether the error was systematic or simply a problem with that single test. Due to the difficulties we had running tests on the Ridge, we were unable to devise and run any program which would conclusively identify such a problem. Further testing is indicated.

3.4. The Hyperbolic Sine Function

In the first two tests, the results were very good, with no significant bit loss evidenced by any of the machines. The final two tests, however, exposed some major problems. While most of the machines tested showed some loss, three of the machines showed almost complete loss of significance: the Onyx, the Pyramid/ATT and the Ridge. The Fortune, Pyramid/BSD, Vax and Sun also showed significant accuracy problems.

Random Accuracy Test #1

Comparison of $\sinh(x)$ *vs.* $\sum_{n=0}^{n=\infty} \frac{x^{2n+1}}{(2n+1)!}$ *in the domain* (0.00000,0.50000)

Machine	Times >	Times =	Times <	Maximum Error	Max. Bit Loss	Root Mean Square	RMS Bit Loss
ATT 3B	1215	2542	1243	2.220007e-16	0.0	1.12602e-16	0
ATT 7300	1102	2787	1111	4.42437e-16	0.994623	1.08322e-16	0
Cyber (VX/VE)	2	2504	2494	7.09275e-15	0.0	3.57747e-15	0
Fortune	1305	2387	1308	4.35089e-16	0.970462	1.23704e-16	0
Masscomp	1327	2390	1283	4.35089e-16	0.970462	1.23604e-16	0
Onyx	1360	2390	1250	4.35674e-16	0.972399	1.2376e-16	0
Pyramid/ATT	48	1139	3813	5.838900e-16	1.394847	1.916810e-16	0
Pyramid/BSD	48	1139	3813	5.838900e-16	1.394847	1.916810e-16	0
Ridge	4	4922	74	2.19636e-16	0.0	1.87059e-17	0
Sun 2	1081	2769	1150	4.42437e-16	0.994623	1.09046e-16	0
Vax	1121	2762	1117	4.55656e-17	0.715166	1.36185e-17	0

Random Accuracy Test #2

Comparison of $\cosh(x)$ *vs.* $\sum_{n=0}^{n=\infty} \frac{x^{2n}}{(2n)!}$ *in the domain* (0.00000,0.50000)

Machine	Times >	Times =	Times <	Maximum Error	Max. Bit Loss	Root Mean Square	RMS Bit Loss
ATT 3B	925	3121	954	2.22045e-16	0.0	1.3077e-16	0
ATT 7300	689	4305	6	2.22045e-16	0.0	7.94084e-17	0
Cyber (VX/VE)	25	2600	2375	7.10543e-15	0.0	4.77928e-15	0
Fortune	229	2536	2235	4.4408e-16	0.999969	1.58108e-16	0
Masscomp	886	3146	968	2.22045e-16	0.0	1.29981e-16	0
Onyx	1086	3139	775	2.22045e-16	0.0	1.30166e-16	0
Pyramid/ATT	0	2330	2670	2.220446e-16	0.0	1.561641e-16	0
Pyramid/BSD	0	6	4994	6.656272e-16	1.583865	3.560114e-16	0.681074
Ridge	3	4490	507	2.21997e-16	0.0	6.78899e-17	0
Sun 2	2500	2488	12	4.41663e-16	0.992096	1.53869e-16	0
Vax	4841	159	0	8.17971e-17	1.55927	3.48799e-17	0.329616

Random Accuracy Test #3

Comparison of $\sinh(x)$ *vs.* $\frac{\sinh(x+1)+\sinh(x-1)}{2\cosh(1)}$ *in the domain* $(3.00,\infty)$

Machine	Times >	Times =	Times <	Maximum Error	Max. Bit Loss	Root Mean Square	RMS Bit Loss
ATT 3B	2432	2348	220	4.35202e-16	0.970835	1.27533e-16	0
ATT 7300	2227	2544	229	4.49925e-16	1.01883	1.21242e-16	0
Cyber (VX/VE)	4926	74	0	3.7732e-14	2.4088	1.87904e-14	1.40301
Fortune	1604	619	2777	4.00704e-14	7.49554	2.79329e-14	6.97497
Masscomp	2429	2425	146	5.89139e-15	4.72969	1.47496e-16	0
Onyx	1561	608	2831	0.615646	51.3002	0.011705	45.5833
Pyramid/ATT	4404	594	2	0.880797	51.816882	0.027853	46.833989
Pyramid/BSD	2007	945	2048	1.08934e-13	8.938389	2.803692e-14	6.980334
Ridge	5000	0	0	0.351946	50.4934	0.351946	50.4934
Sun 2	1492	572	2936	4.01248e-14	7.4975	2.79358e-14	6.97512
Vax	1563	343	3094	9.27743e-16	5.06288	3.76778e-16	3.76287

Random Accuracy Test #4

Comparison of $\cosh(x)$ *vs.* $\frac{\cosh(x+1)+\cosh(x-1)}{2\cosh(1)}$ *in the domain* $(3.00,\infty)$

Machine	Times >	Times =	Times <	Maximum Error	Max. Bit Loss	Root Mean Square	RMS Bit Loss
ATT 3B	2429	2342	229	4.19221e-16	0.91686	1.2795e-16	0
ATT 7300	2265	2515	220	4.48286e-16	1.01357	1.1991e-16	0
Cyber (VX/VE)	4930	70	0	3.8618e-14	2.44228	1.86952e-14	1.39568
Fortune	1551	645	2804	4.00386e-14	7.4944	2.79197e-14	6.97429
Masscomp	2449	2407	144	4.12332e-16	0.892958	1.22777e-16	0
Onyx	1582	614	2804	0.662282	51.4055	0.0125933	45.6888
Pyramid/ATT	4401	595	4	0.880797	51.816882	0.027853	46.833989
Pyramid/BSD	2069	936	1995	3.989801e-14	7.489324	2.790485e-14	6.973522
Ridge	5000	0	0	0.351946	50.4934	0.351946	50.4934
Sun 2	1487	555	2958	4.00396e-14	7.49443	2.78918e-14	6.97285
Vax	1628	344	3028	9.08458e-16	5.03257	3.84755e-16	3.79309

The Cyber raised multitudes of FPE (floating point exception/error) signals in the special tests at the end to check for overflow/underflow errors, and we were unable to establish why that condition was being signalled for values that (theoretically) should have been in range. The other systems returned results within expected ranges.

3.5. The Hyperbolic Tangent Function

Here again the Masscomp showed significant accuracy problems. The other machines make very reasonable showings. In the second test it was the Fortune which showed somewhat poor results, but the RMS loss was not so significant as to cause excessive concern. The other systems all produced near-perfect results.

Random Accuracy Test #1

Comparison of $\tanh(x)$ *vs.*

$$\frac{\tanh\left[x-\frac{1}{8}\right]+\tanh\left[\frac{1}{8}\right]}{1+\tanh\left[x-\frac{1}{8}\right]\times\tanh\left[\frac{1}{8}\right]}$$

in the domain (0.125,0.549306)

Machine	Times >	Times =	Times <	Maximum Error	Max. Bit Loss	Root Mean Square	RMS Bit Loss
ATT 3B	1224	2792	984	2.99258e-16	0.430541	1.02812e-16	0
ATT 7300	1773	1740	1487	6.09293e-16	1.45629	1.73833e-16	0
Cyber (VX/VE)	4619	373	8	3.53048e-14	2.31287	1.08594e-14	0.611948
Fortune	2896	1238	866	8.82621e-16	1.99094	2.37577e-16	0.0975441
Masscomp	1731	1691	1578	1.58501e-10	19.4452	2.24155e-12	13.3014
Onyx	1678	1736	1586	6.20157e-16	1.48178	1.76113e-16	0
Pyramid/ATT	1705	1730	1565	6.940739e-16	1.64424	1.762941e-16	0
Pyramid/BSD	2581	1285	1134	8.231316e-16	1.890274	2.292866e-16	0.046302
Ridge	1613	2693	694	4.39925e-16	0.986408	1.08161e-16	0
Sun 2	909	1178	2913	8.95365e-16	2.01163	2.49867e-16	0.17031
Vax	867	991	3142	1.16407e-16	2.06833	3.4569e-17	0.316702

Random Accuracy Test #2

Comparison of $\tanh(x)$ *vs.*

$$\frac{\tanh\left[x-\frac{1}{8}\right]+\tanh\left[\frac{1}{8}\right]}{1+\tanh\left[x-\frac{1}{8}\right]\times\tanh\left[\frac{1}{8}\right]}$$

in the domain (0.674306,19.061547)

Machine	Times >	Times =	Times <	Maximum Error	Max. Bit Loss	Root Mean Square	RMS Bit Loss
ATT 3B	1566	2223	1211	3.56547e-16	0.683242	9.4157e-17	0
ATT 7300	1844	1526	1630	4.4413e-16	1.00013	1.46571e-16	0
Cyber (VX/VE)	3837	1048	115	1.49262e-14	1.07085	5.56637e-15	0
Fortune	1781	1427	1792	7.10543e-15	5	5.11158e-16	1.20292
Masscomp	1917	1506	1577	4.4413e-16	1.00013	1.47735e-16	0
Onyx	1564	2343	1093	3.67494e-16	0.72687	9.43772e-17	0
Pyramid/ATT	2207	1566	1227	5.551115e-16	1.321928	1.532602e-16	0
Pyramid/BSD	2198	1558	1244	5.558423e-16	1.323826	1.532351e-16	0
Ridge	1442	2253	1305	3.62954e-16	0.708936	9.27002e-17	0
Sun 2	2147	1553	1300	5.55112e-16	1.32193	1.47131e-16	0
Vax	1372	1509	2119	7.09125e-17	1.35326	1.89882e-17	0

3.6. The Logarithm Function

The first test showed significant loss on the Masscomp and the Cyber, but the rest of the machines were acceptable with minimal bit loss. The Masscomp lost nearly its entire range of precision, with an RMS loss of over 45 bits out of 52. The Cyber results were also distressing, with an RMS bit loss of over 18.

Random Accuracy Test #1

Comparison of $\ln(x)$ *vs.* $-\sum_{i=1}^{i=4}\frac{(1-x)^i}{i}$ *in the domain* $(1-\epsilon,1+\epsilon)$

Machine	Times >	Times =	Times <	Maximum Error	Max. Bit Loss	Root Mean Square	RMS Bit Loss
ATT 3B	1035	2949	1016	3.29178e-16	0.568018	1.009e-16	0
ATT 7300	1723	2923	354	3.53493e-16	0.670833	1.02159e-16	0
Cyber (VX/VE)	2758	920	1322	1.33118e-07	24.1592	2.59343e-09	18.4775
Fortune	1337	2210	1453	3.81103e-15	4.10126	2.570737e-16	0.175323
Masscomp	1	0	4999	0.896581	51.8425	0.0126796	45.6987
Onyx	1289	2215	1496	4.52927e-16	1.02843	1.33912e-16	0
Pyramid/ATT	1541	2993	466	2.220271e-16	0.0	9.904831e-17	0
Pyramid/BSD	3662	1182	156	6.082408e-16	1.453793	1.949612e-16	0
Ridge	1723	2923	354	3.53493e-16	0.670833	1.02159e-16	0
Sun 2	2062	2180	758	5.33434e-16	1.26446	1.38228e-16	0
Vax	10	319	4671	8.19698e-17	1.56231	3.56e-17	0.359101
Ours	1658	3025	317	4.42096e-17	0.671582	1.23752e-17	0

The results of the following three tests were acceptable in over half of the cases, but some of the results, especially those on the Vax, Onyx, Sun, Pyramid/BSD and Fortune are much higher than they should be for this function.

Random Accuracy Test #2

Comparison of $\ln(x)$ *vs.* $\ln\left[\frac{17x}{16}\right]-\ln\left[\frac{17}{16}\right]$ *in the domain* $(\frac{1}{\sqrt{2}},\frac{15}{16})$

Machine	Times >	Times =	Times <	Maximum Error	Max. Bit Loss	Root Mean Square	RMS Bit Loss
ATT 3B	720	3002	1278	4.31714e-16	0.959228	1.071e-16	0
ATT 7300	533	3235	1232	4.02682e-16	0.858793	9.99455e-17	0
Cyber (VX/VE)	520	1617	2863	6.18928e-14	3.12278	1.89954e-14	1.41866
Fortune	375	900	3725	2.43412e-15	3.45448	8.40023e-16	1.91958
Masscomp	1311	1246	2443	7.35294e-16	1.72747	2.84988e-16	0.360053
Onyx	393	922	3685	2.43412e-15	3.45448	8.37585e-16	1.91539
Pyramid/ATT	1038	3036	926	3.555303e-16	0.679123	1.064681e-16	0
Pyramid/BSD	54	542	4404	2.271627e-15	3.354804	8.069028e-16	1.861545
Ridge	531	3237	1232	4.02682e-16	0.858793	9.97687e-17	0
Sun 2	291	886	3823	2.43287e-15	3.45374	8.36598e-16	1.91369
Vax	415	336	4249	2.14352e-15	6.27106	7.92616e-16	4.83577
Ours	633	3158	1209	4.42375e-17	0.672493	1.29621e-17	0

Random Accuracy Test #3 *Comparison of* $\log_{10}(x)$ *vs.* $\log_{10}\left[\frac{11x}{10}\right]-\log_{10}\left[\frac{11}{10}\right]$ *in the domain* $(\frac{1}{\sqrt{10}},0.900000)$							
Machine	Times >	Times =	Times <	Maximum Error	Max. Bit Loss	Root Mean Square	RMS Bit Loss
ATT 3B	1728	1924	1348	6.12079e-16	1.46287	1.68061e-16	0
ATT 7300	2047	1886	1067	7.53558e-16	1.76287	1.76514e-16	0
Cyber (VX/VE)	534	1020	3446	7.40787e-14	3.38206	1.83706e-14	1.37041
Fortune	1471	842	2687	4.41346e-15	4.31299	1.32033e-15	2.57197
Masscomp	1677	2239	1084	9.49037e-16	2.09562	1.81603e-16	0
Onyx	1350	835	2815	4.24089e-15	4.25545	1.31048e-15	2.56117
Pyramid/ATT	422	752	3826	1.439413e-15	2.696559	4.012940e-16	0.85381
Pyramid/BSD	871	474	3655	4.421482e-15	4.315609	1.418431e-15	2.675375
Ridge	1642	1941	1417	6.65911e-16	1.58488	1.67815e-16	0
Sun 2	1353	874	2773	4.59736e-15	4.37188	1.39596e-15	2.65234
Vax	986	448	3566	4.0565e-15	7.19131	1.3071e-15	5.55745
Ours	1538	1910	1552	7.73536e-17	1.47869	2.10383e-17	0

Random Accuracy Test #4 *Comparison of* $\ln(x^2)$ *vs.* $2\ln(x)$ *in the domain* (16.00000,240.0000)							
Machine	Times >	Times =	Times <	Maximum Error	Max. Bit Loss	Root Mean Square	RMS Bit Loss
ATT 3B	91	4801	108	2.20843e-16	0.0	3.37023e-17	0
ATT 7300	65	4808	127	2.21831e-16	0.0	3.33234e-17	0
Cyber (VX/VE)	1172	2331	1497	1.26099e-14	0.827568	4.2532e-15	0
Fortune	1019	3058	923	4.29034e-16	0.950242	1.15952e-16	0
Masscomp	77	4817	106	2.2017e-16	0.0	3.25838e-17	0
Onyx	1030	3059	911	2.91761e-13	10.3597	4.12775e-15	4.21644
Pyramid/ATT	51	4548	401	2.205582e-16	0.0	4.948310e-17	0
Pyramid/BSD	918	2874	1208	5.688133e-16	1.357106	1.247391e-16	0
Ridge	260	4587	153	2.17983e-16	0.0	4.34932e-17	0
Sun 2	985	2963	1052	4.29034e-16	0.950242	1.19437e-16	0
Vax	1863	1261	1876	3.74052e-16	3.75239	9.15732e-17	1.72215
Ours	52	4848	100	2.74424e-17	0	3.65939e-18	0

In special testing, the Onyx did not signal errors when presented with negative arguments (for which the function is undefined), but returned the value zero. We view this as a serious problem. Nothing else unusual was noted in the special tests.

3.7. The Power Function

The C language does not have an intrinsic function to raise values to a real power. Most C runtime libraries provide such a function, named *pow* or *power*. It is this function which we tested in this section.

There were no poor showings in the first two tests, except for the Vax which was off by a few bits. In the last two tests, there were no acceptable results, with the least significant loss being almost five bits and three machines showing a loss of all bits of precision in the worst case. The Ridge machine showed the worst results on these tests, with the Onyx and Pyramid/ATT not far behind.

We took special care in testing the code for these tests because the results seemed so uniformly poor. However, the comparisons are extremely simple (as can be seen from the identities used), and verification of some of the worst-point errors by calculator did indeed show that the machines in question were performing as badly as the numbers seem to indicate.

When we implemented the routine ourselves and ran the same tests on it (producing the values shown) we came to the conclusion that the test is accurate and the standard implementations on all of the machines tested are faulty.

Random Accuracy Test #1
Comparison of $x^{1.0}$ *vs.* x *in the domain* (0.500,1.000)

Machine	Times >	Times =	Times <	Maximum Error	Max. Bit Loss	Root Mean Square	RMS Bit Loss
ATT 3B	0	5000	0	0	0	0	0
ATT 7300	22	4713	265	1.89086e-16	0	3.59852e-17	0
Cyber (VX/VE)	0	5000	0	0	0	0	0
Fortune	2437	1416	1147	1.41834e-15	2.67528	3.71491e-16	0.743476
Masscomp	0	5000	0	0	0	0	0
Onyx	0	5000	0	0	0	0	0
Pyramid/ATT	1414	3482	104	3.453348e-16	0.637146	1.007427e-16	0
Pyramid/BSD	3248	1020	732	1.574116e-15	2.82562	3.957481e-16	0.833733
Ridge	27	4692	281	1.94277e-16	0.0	4.05047e-17	0
Sun 2	1871	740	2389	1.57364e-15	2.82519	4.28709e-16	0.949149
Vax	1134	363	3503	1.23592e-15	5.47666	3.53055e-16	3.66905
Ours	0	5000	0	0	0	0	0

Random Accuracy Test #2
Comparison of $\left[x \times x\right]^{1.5}$ *vs.* x^3 *in the domain* (0.500,1.000)

Machine	Times >	Times =	Times <	Maximum Error	Max. Bit Loss	Root Mean Square	RMS Bit Loss
ATT 3B	1184	2992	824	4.41714e-16	0.992263	1.10686e-16	0
ATT 7300	1126	3015	859	4.41714e-16	0.992263	1.1135e-16	0
Cyber (VX/VE)	947	1417	2636	2.51725e-14	1.82486	6.6744e-15	0
Fortune	2348	1177	1475	2.24147e-15	3.33552	5.50666e-16	1.31033
Masscomp	1257	3005	738	4.41714e-16	0.992263	1.08893e-16	0
Onyx	1206	1073	2721	2.23987e-15	3.3345	5.52641e-16	1.31549
Pyramid/ATT	410	2010	2580	6.609585e-16	1.57371	1.777601e-16	0
Pyramid/BSD	1583	567	2850	2.463930e-15	3.47204	6.472624e-16	1.543501
Ridge	739	3052	1209	4.40639e-16	0.988748	1.08138e-16	0
Sun 2	2892	492	1616	2.65176e-15	3.57803	6.81945e-16	1.6188
Vax	1586	589	2825	1.79656e-15	6.01632	5.30098e-16	4.25541
Ours	1306	3422	272	2.77536e-17	0	1.14913e-17	0

Random Accuracy Test #3 *Comparison of* $\left[x \times x\right]^{1.5}$ *vs.* x^3 *in the domain* (1.000,∞)							
Machine	Times >	Times =	Times <	Maximum Error	Max. Bit Loss	Root Mean Square	RMS Bit Loss
ATT 3B	2561	7	2432	9.92374e-14	8.80389	4.33243e-14	7.60818
ATT 7300	1889	8	3103	9.94043e-14	8.80632	4.22514e-14	7.57201
Cyber (VX/VE)	2511	2	2487	1.09131e-11	10.5848	5.01247e-12	9.46238
Fortune	4245	3	752	2.10762e-13	9.89055	7.78419e-14	8.45355
Masscomp	2568	7	2425	9.9372e-14	8.80585	4.33425e-14	7.60879
Onyx	2877	6	2117	1.76219	52	0.553395	51.1464
Pyramid/ATT	4998	2	0	1.0	52	0.454313	50.861757
Pyramid/BSD	5000	0	0	5.528587e-13	11.281846	3.147988e-13	10.469365
Ridge	5000	0	0	1.0	52	1.0	52
Sun 2	0	0	5000	3.53297e-13	10.6358	2.29011e-13	10.0104
Vax	538	30	4432	5.01713e-15	7.49794	1.48756e-15	5.74403
Ours	1270	3465	265	2.76327e-17	0	1.12559e-17	0

Random Accuracy Test #4 *Comparison of* x^y *vs.* $(x^2)^{\left[\frac{y}{2}\right]}$ *in the domain* (0.0100,10.0)							
Machine	Times >	Times =	Times <	Maximum Error	Max. Bit Loss	Root Mean Square	RMS Bit Loss
ATT 3B	293	4448	259	1.13721e-13	9.00043	6.80548e-15	4.93778
ATT 7300	244	4495	261	1.13675e-13	8.99985	6.45163e-15	4.86074
Cyber (VX/VE)	1185	2656	1159	1.45563e-11	11.0004	2.57417e-12	8.50097
Fortune	1347	2337	1316	2.6643e-13	10.0576	3.78481e-14	7.41323
Masscomp	375	4272	353	1.13764e-13	9.00098	7.53427e-15	5.08455
Onyx	1367	2307	1326	2.36394e-13	10.0561	3.58712e-14	7.33583
Pyramid/ATT	612	3763	625	1.137895e-13	9.001302	1.026457e-14	5.53068
Pyramid/BSD	1360	2304	1336	1.973562e-13	9.795737	3.738688e-14	7.395539
Ridge	2474	2526	0	1.0	52	0.70342	51.4925
Sun 2	1469	2102	1429	2.36516e-13	10.0569	4.11764e-14	7.53482
Vax	1818	1356	1826	2.27545e-14	9.67916	4.31078e-15	7.27903
Ours	387	4241	372	3.75841e-17	0.437343	7.68458e-18	0

The other special tests revealed nothing of particular interest.

3.8. The Sine Function

The Onyx showed the only significant loss in the first test, with five of the machines showing significant loss in the second and third tests. Some of the machines performed incredibly well, with RMS bit losses of zero in all three tests. This makes the problems with the others (particularly the Onyx) especially notable.

Random Accuracy Test #1

Comparison of $\sin(x)$ *vs.* $3\sin\left[\frac{x}{3}\right]-4\sin^3\left[\frac{x}{3}\right]$ *in the domain* $\left[0.00000,\frac{\pi}{2}\right]$

Machine	Times >	Times =	Times <	Maximum Error	Max. Bit Loss	Root Mean Square	RMS Bit Loss
ATT 3B	826	3104	1070	3.34765e-16	0.592298	8.86461e-17	0
ATT 7300	773	3150	1077	2.63705e-16	0.248076	8.71758e-17	0
Cyber (VX/VE)	2014	2442	544	1.43487e-14	1.01392	3.87677e-15	0
Fortune	1691	1884	1425	6.58201e-16	1.56768	1.53307e-16	0
Masscomp	925	3196	879	2.72793e-16	0.296956	8.48542e-17	0
Onyx	1580	1889	1531	9.5859e-11	18.7197	1.35565e-12	12.5758
Pyramid/ATT	4529	471	0	5.551168e-16	1.321942	2.232843e-16	0.008032
Pyramid/BSD	4142	713	145	7.794822e-16	1.811667	2.944877e-16	0.407358
Ridge	659	2794	1547	3.38603e-16	0.608745	1.02594e-16	0
Sun 2	991	1712	2297	5.89939e-16	1.40972	1.66725e-16	0
Vax	3040	1241	719	9.71862e-17	1.80797	3.02549e-17	0.12439
Ours	929	3085	986	3.64065e-17	0.391419	1.10045e-17	0

Random Accuracy Test #2

Comparison of $\sin(x)$ *vs.* $3\sin\left[\frac{x}{3}\right]-4\sin^3\left[\frac{x}{3}\right]$ *in the domain* $\left[6\pi,\frac{13\pi}{2}\right]$

Machine	Times >	Times =	Times <	Maximum Error	Max. Bit Loss	Root Mean Square	RMS Bit Loss
ATT 3B	1093	2476	1431	4.02569e-16	0.858386	1.11415e-16	0
ATT 7300	1008	2525	1467	4.02569e-16	0.858386	1.10712e-16	0
Cyber (VX/VE)	2201	2382	417	1.42404e-14	1.00299	3.88802e-15	0
Fortune	2144	714	2142	9.2727e-13	12.0279	3.4184e-14	7.26633
Masscomp	949	3096	955	2.76188e-15	3.63673	9.54463e-17	0
Onyx	2189	710	2101	3.79094e-11	17.3813	5.39942e-13	11.2477
Pyramid/ATT	4193	793	14	5.561104e-16	1.324522	2.188522e-16	0
Pyramid/BSD	4003	240	757	3.790906e-11	17.381334	5.397371e-13	11.247192
Ridge	975	2301	1724	4.08332e-16	0.878892	1.19855e-16	0
Sun 2	1795	661	2544	9.2727e-16	12.0279	3.41833e-14	7.2663
Vax	2839	528	1633	9.47731e-12	18.3813	1.3443e-13	12.2418
Ours	1094	2786	1120	4.53451e-17	0.70817	1.25078e-17	0

Random Accuracy Test #3 *Comparison of* $\cos(x)$ *vs.* $4\cos\left[\frac{x}{3}\right]^3 - 3\cos\left[\frac{x}{3}\right]$ *in the domain* $\left[7\pi, \frac{15\pi}{2}\right]$							
Machine	Times >	Times =	Times <	Maximum Error	Max. Bit Loss	Root Mean Square	RMS Bit Loss
ATT 3B	783	2484	1733	4.07055e-16	0.874374	1.11319e-16	0
ATT 7300	720	2537	1743	4.07055e-16	0.874374	1.08335e-16	0
Cyber (VX/VE)	1987	2068	945	1.40852e-14	0.987183	4.26148e-15	0
Fortune	2202	722	2076	7.20247e-12	14.9854	1.26519e-13	9.15428
Masscomp	910	3170	920	4.19798e-16	0.918845	8.63866e-17	0
Onyx	2103	723	2174	1.44051e-11	15.9854	2.77779e-13	10.2889
Pyramid/ATT	4215	769	16	5.561869e-16	1.32472	2.257501e-16	0.023877
Pyramid/BSD	2470	323	2207	4.597780e-12	14.3378	9.104983e-14	8.679663
Ridge	958	2434	1608	4.25444e-16	0.938121	1.15762e-16	0
Sun 2	1865	652	2483	7.20261e-12	14.9854	1.26519e-13	9.15429
Vax	2743	541	1716	5.747e-13	14.3377	1.16884e-14	8.71808
Ours	1030	2951	1019	4.8341e-17	0.800471	1.1711e-17	0

3.9. The Square Root Function

The square root function is one of the simplest functions to implement. Using a modified Newton's method, the square root function should be both fast and highly accurate in any implementation, as the following test results seem to indicate. Worth noting: the Masscomp showed the best performance on this test!

Random Accuracy Test #1 *Comparison of* $\sqrt{x^2}$ *vs.* x *in the domain* $(\frac{1}{\sqrt{2}}, 1.00000)$							
Machine	Times >	Times =	Times <	Maximum Error	Max. Bit Loss	Root Mean Square	RMS Bit Loss
ATT 3B	195	4594	211	1.56933e-16	0	3.97318e-17	0
ATT 7300	385	4615	0	1.56895e-16	0	3.89012e-17	0
Cyber (VX/VE)	0	0	5000	5.02406e-15	0	4.25992e-15	0
Fortune	187	4612	201	1.56933e-16	0	3.91771e-17	0
Masscomp	0	5000	0	0	0	0	0
Onyx	229	4619	152	1.56933e-16	0	3.8532e-17	0
Pyramid/ATT	0	0	5000	1.570018e-16	0	1.331226e-16	0
Pyramid/BSD	0	0	5000	1.570018e-16	0	1.331226e-16	0
Ridge	264	4736	0	1.56895e-16	0	3.27018e-17	0
Sun 2	401	4599	0	1.56954e-16	0	3.96277e-17	0
Vax	394	4606	0	1.96192e-17	0	4.95469e-18	0

Random Accuracy Test #2 *Comparison of* $\sqrt{x^2}$ *vs.* x *in the domain* $(1.000,\sqrt{2})$							
Machine	Times >	Times =	Times <	Maximum Error	Max. Bit Loss	Root Mean Square	RMS Bit Loss
ATT 3B	0	5000	0	0	0	0	0
ATT 7300	0	5000	0	0	0	0	0
Cyber (VX/VE)	0	0	5000	7.10538e-15	0	6.02606e-15	0
Fortune	0	5000	0	0	0	0	0
Masscomp	0	5000	0	0	0	0	0
Onyx	11	4989	0	2.21757e-16	0	9.94302e-18	0
Pyramid/ATT	0	0	5000	2.220430e-16	0	1.883144e-16	0
Pyramid/BSD	0	0	5000	2.220430e-16	0	1.883144e-16	0
Ridge	0	3587	1413	2.21928e-16	0	9.7865e-17	0
Sun 2	0	5000	0	0	0	0	0
Vax	0	5000	0	0	0	0	0

3.10. The Tangent Function

All of the following results are quite acceptable, although we believe that the maximum bit error could be reduced to one bit or under with some extra care in the coding.

Random Accuracy Test #1 *Comparison of* $\tan(x)$ *vs.* $\dfrac{2\tan\left[\frac{x}{2}\right]}{1-\tan^2\left[\frac{x}{2}\right]}$ *in the domain* $(0.00,\frac{\pi}{4})$							
Machine	Times >	Times =	Times <	Maximum Error	Max. Bit Loss	Root Mean Square	RMS Bit Loss
ATT 3B	1421	2129	1450	5.66419e-16	1.35102	1.45357e-16	0
ATT 7300	2822	1497	681	7.19545e-16	1.69623	1.96076e-16	0
Cyber (VX/VE)	1436	1852	1712	1.91678e-14	1.43169	5.3201e-15	0
Fortune	1475	1884	1641	6.61098e-16	1.57402	1.64873e-16	0
Masscomp	1443	1931	1626	6.67475e-16	1.58786	1.61357e-16	0
Onyx	1854	1843	1303	6.61098e-16	1.57402	1.69071e-16	0
Pyramid/ATT	594	1463	2943	8.205748e-16	1.885785	2.060405e-16	0
Pyramid/BSD	594	1463	2943	8.205748e-16	1.885785	2.060405e-16	0
Ridge	2895	1828	277	5.05286e-16	1.18625	1.5701e-16	0
Sun 2	2787	1528	685	6.61098e-16	1.57402	1.95144e-16	0
Vax	2385	1692	923	8.19926e-17	1.56272	2.25124e-17	0

Random Accuracy Test #2

Comparison of $\tan(x)$ *vs.* $\dfrac{2\tan\left[\frac{x}{2}\right]}{1-\tan^2\left[\frac{x}{2}\right]}$ *in the domain* $(\frac{7\pi}{8},\frac{9\pi}{8})$

Machine	Times >	Times =	Times <	Maximum Error	Max. Bit Loss	Root Mean Square	RMS Bit Loss
ATT 3B	1321	2468	1211	4.43273e-16	0.997345	1.28405e-16	0
ATT 7300	1616	1852	1532	5.88609e-16	1.40646	1.62415e-16	0
Cyber (VX/VE)	191	1160	3649	2.35253e-14	1.727222	7.30308e-15	0.0395834
Fortune	1515	1883	1602	6.04283e-16	1.44438	1.6373r-16	0
Masscomp	1514	1903	1583	6.62434e-16	1.57693	1.63701e-16	0
Onyx	1597	1931	1472	6.6018e-16	1.57201	1.60875e-16	0
Pyramid/ATT	577	1492	2931	8.098918e-16	1.86688	2.030279e-16	0
Pyramid/BSD	577	1492	2931	8.098918e-16	1.86688	2.030279e-16	0
Ridge	1182	2781	1027	4.19255e-16	0.91698	1.11753e-16	0
Sun 2	1577	1871	1552	6.62434e-16	1.57693	1.65108e-16	0
Vax	1524	1931	1545	7.63425e-17	1.45971	2.01501e-17	0

Random Accuracy Test #3

Comparison of $\tan(x)$ *vs.* $\dfrac{2\tan\left[\frac{x}{2}\right]}{1-\tan^2\left[\frac{x}{2}\right]}$ *in the domain* $(6\pi,\frac{25\pi}{4})$

Machine	Times >	Times =	Times <	Maximum Error	Max. Bit Loss	Root Mean Square	RMS Bit Loss
ATT 3B	1353	2189	1458	5.20375e-16	1.2287	1.42333e-16	0
ATT 7300	2802	1530	668	6.93358e-16	1.64275	1.95984e-16	0
Cyber (VX/VE)	1453	1827	1720	1.78734e-14	1.33082	5.3272e-15	0
Fortune	1565	1861	1574	6.31663e-16	1.5083	1.63068e-16	0
Masscomp	1443	1869	1688	6.37238e-16	1.52098	1.64341e-16	0
Onyx	1867	1851	1282	6.14785e-16	1.46923	1.65055e-16	0
Pyramid/ATT	632	1382	2986	6.305834e-16	1.505838	2.066444e-16	0
Pyramid/BSD	632	1382	2986	6.305834e-16	1.505838	2.066444e-16	0
Ridge	2944	1804	252	4.8177e-16	1.11749	1.60248e-16	0
Sun 2	2784	1528	688	6.92056e-16	1.64004	1.95036e-16	0
Vax	2453	1632	915	1.06223e-16	1.93624	2.28498e-17	0

Random Accuracy Test #4

Comparison of $cot(x)$ *vs.* $\frac{cot^2\left[\frac{x}{2}\right]-1}{2cot\left[\frac{x}{2}\right]}$ *in the domain* $(6\pi,\frac{25\pi}{4})$

Machine	Times >	Times =	Times <	Maximum Error	Max. Bit Loss	Root Mean Square	RMS Bit Loss
ATT 3B	1464	2105	1431	5.93035e-16	1.41727	1.55995e-16	0
ATT 7300	1336	1915	1749	6.88357e-16	1.63231	1.71357e-16	0
Cyber (VX/VE)	3468	1189	343	2.64962e-14	1.89879	8.05104e-15	0.180254
Fortune	1574	1915	1511	7.63499e-16	1.78178	1.71601e-16	0
Masscomp	1604	1871	1525	6.39152e-16	1.52531	1.74369e-16	0
Onyx	1318	1815	1867	6.60695e-16	1.57313	1.763e-16	0
Pyramid/ATT	3688	1031	281	7.544874e-16	1.764647	2.594292e-16	0.224491
Pyramid/BSD	3688	1031	281	7.544874e-16	1.764647	2.594292e-16	0.224491
Ridge	1101	2498	1401	4.73652e-16	1.09298	1.28884e-16	0
Sun 2	1437	1860	1703	6.66121e-16	1.58493	1.73726e-16	0
Vax	1720	1845	1435	7.5468e-17	1.44309	2.20386e-17	0

4. Error Indications

Both BSD and AT&T UNIX have a standard floating point error signal defined — SIGFPE — but they use the signal differently. The AT&T implementations all use the signal to indicate that a domain error has occurred (such as trying to take the square root of a negative number), or that a range error had occurred (such as overflow). It is relatively simple to catch these signals and report errors.

Under BSD UNIX, however, the SIGFPE error is not signalled automatically by the math routines if they fail. Instead, those routines set an external error variable which the user program must test, and they return a value (possibly 0). A user not aware of this convention might very well use the values returned, leading to bizarre and unexplained results. Under the BSD implementation, the FPE signal seems to be reserved for hardware-detected floating point faults only (*e.g.,* divide by zero or subscript out-of-range).

An interesting difference to this is the Sun, which runs a BSD UNIX environment. Errors do not result in signals, but result in the return values being set (as in the IEEE 754 draft standard) to values like *infinity* and *not-a-number.* These values can be used in further calculations in a replicable and well-defined way. Although the user might not get immediate notification that an error has occurred, the end results will reflect the error.

In our tests, the Cyber and the Ridge signalled errors in an undocumented but replicable way — they went into permanent error states, sometimes printing (continuously) error messages to the screen. The usual cure on both systems was to abort the program, but more than one error on the Ridge required the machine to be rebooted to recover. This is unfortunate because the Ridge documents a very flexible and well-designed error handling interface. However, we could not always get it to work the way it was documented, so we cannot comment further on its behavior other than to say that user code could not reliably depend on it.

We should note that some of the machines failed to return errors when presented with arguments that were out of the domain of the implementation.[22] We also noted cases where absolutely no error was indicated when the function was presented with an argument for which the

[22] The result would be too small or large to be properly represented (range error), or too much precision would be lost in the attempt at calculating a value, even though the value is within the defined domain for the actual mathematical function.

function was undefined. Since we had difficulties getting the software to perform correctly on all the machines during these tests, we have not noted the majority of these cases in this report. We would urge anyone running their own accuracy tests to be sure to check the behavior of their functions for improper and out-of-range input values.

5. Other Interesting Results

We discovered a number of interesting problems in porting these tests to the various machines and running them. Some of the interesting things we discovered are presented here.

We had incredible problems with supposedly standard UNIX features on the Ridge and on the Cyber. Both had problems with signal handling and I/O redirection. We were unable to capture error output in the same file as the results on either system, although this is simple to do under standard UNIX systems. Both systems had buggy[23] implementations of the signal handling libraries and this resulted in many of the tests going into infinite loops. Both systems required substantial editing of the programs to get them to run in an acceptable manner.

The Masscomp compiler had an option for compiling in special code on machines which were not equipped with a floating point accelerator. We compiled all of our tests using that option.[24] The code included by the compiler in this case does the floating point operations in software as inline code or subroutines, we assume. After running the tests, we tried compiling all of the tests again **without** the option, just as if there was an FPA present, to see what would happen. As we expected, the tests ran somewhat slower. In this case, we assume that the instructions that would normally be executed by a floating point board were instead being trapped and emulated by either microcode or operating system-level software routines. What surprised us, however, was the fact that the test results were **measurably and significantly less accurate** under these conditions. We found it especially distressing that there was no warning or indication in the software that there was no floating point hardware available, and that (evidently, less accurate) software methods were being used instead.

We found that on the Pyramid, a value of minus zero was not equal to a positive zero. We are not sure how frequently code might result in a value of "minus zero," but comparisons against a constant zero will sometimes produce misleading results. This appears to be a bug in the floating point hardware of the Pyramid, although since this bug was reported to Pyramid, we have been unable to reproduce this condition under the current release of the software and we assume that some corrective action has been taken, at least on our machine.

Even excepting the Ridge and the Cyber, we had to produce four different versions of our mainline test driver to handle peculiarities in the different versions of UNIX used on these machines. There were problems with output buffering, unusual compiler options, differences in library calls and definition files, differences in behavior of copy and edit programs, and many other subtle, but often critical, differences. UNIX is sometimes claimed to be an operating system enhancing portability of software. That statement is true to only a limited degree. The differences in signal and error handling, especially, lead to some major portability problems. Code taken from a System V environment to a BSD environment, for example, might be run without the realization that errors will no longer trigger an FPE signal, thus resulting in undetected errors.

6. Conclusions

We can draw a number of conclusions from this work, many of which can be extended to other systems and programming environments. First of all, we believe it to be very important that anyone wishing to perform critical floating point work under a UNIX or UNIX-like environment perform some form of testing and verification that their systems are equal to the expected tasks. The only versions of the math libraries producing consistently good results in our tests were those running on the AT&T machines, and they performed poorly in two of the *power* function tests. We showed with our routines that (on at least one machine) there are no reasons why accurate implementations cannot

[23] In our estimation.

[24] As noted earlier, the Masscomp we tested was not equipped with a floating point accelerator.

be obtained. At the very least, the standard routines could be tested and documented as to the algorithms involved and the errors produced.

We are particularly disturbed by thoughts of critical components of items such as nuclear reactors, weapons control systems, critical health care equipment or aircraft being designed using software written in C on any of these systems and depending on the standard libraries. Errors can cascade and multiply quickly, and with routines as inaccurate as some of the ones we tested, it would be very easy to produce results significantly in error.

Our results would also cause us to closely question any research results which depend on these (or related) library functions and which do not provide some verification of their underlying calculations. This is of considerable concern, especially considering the normal market where these machines are sold.

A second conclusion is that any mathematical software should be written carefully with error detection in mind. Errors are signalled and dealt with in what seems to be a different manner for each machine despite their common software origins (UNIX). In some implementations, it is possible for error indications to be missed, resulting in propagation of data severely in error into final results. We hope that the development of the IEEE floating-point standard is going to be a major aid in this area; we were especially impressed with the way elements of the draft standard have been integrated into the hardware and software of the Sun.

A third conclusion is well-known to many people but bears restating here: many systems claiming to be UNIX or UNIX-like quite simply **are not.** Neither are programs completely portable from one UNIX system to another. For instance, the amount of frustration and energy spent trying to make simple UNIX features (such as I/O redirection) work on two of the systems claiming to implement UNIX was more than was spent in coding and debugging most of the tests on the other systems. Despite some vendor claims, we do not currently recognize any particular version of UNIX as "standard," in part because of the number and popularity of the different versions of UNIX in existence, and in part due to the different requirements and environments those versions are expected to fill. Our experiences in writing and running these tests has convinced us that such a standard, would certainly be desirable, however. The current efforts of the IEEE/P1003 standards committee may address some of these concerns.

In closing, we should note that although C is a very powerful and general programming language, we view it as less than ideal for many mathematical and scientific applications. Our experience with these tests leads us to believe that some implementations of the C library serve to hinder the derivation of accurate mathematical results, rather than support such efforts. Unfortunately, there appears to be no other language currently available under UNIX (to our knowledge) which has the power and flexibility of C, and which is better suited for mathematical applications. Our experience in implementing some selected functions in C did, however, reaffirm the ease with which complex software can be easily coded in this language; there is little doubt in our minds that the task would have been (almost) impossible to accomplish in any "safe" language such as Pascal or Cobol.

Finally, let us note, with some small amount of regret: *Caveat emptor.*

7. Acknowledgements

We would like to gratefully acknowledge the assistance of the following people in running our tests and battling unfamiliar machines: Don Deal, Office of Computing Services, Georgia Tech; Terry Countryman, Medical Systems Development Corporation, Atlanta; Charles Cleveland, School of Physics, Georgia Tech; Jeff Lee, School of Information and Computer Science, Georgia Tech; Carter Bullard, School of Information and Computer Science, Georgia Tech; A. Jeff Offutt, VI, School of Information and Computer Science, Georgia Tech; W. Ken Thompson, Georgia Tech Research Institute; and Peter Wan, Computer Systems Resources, Inc., Atlanta. We would also like to thank David Pitts for running into the original problem which prompted this report, and Dana Eckart for helping us identify the "negative zero" problem on the Pyramid.

References

Bour83.
S. R. Bourne, *The UNIX System,* Addison-Wesley, Reading, MA, 1983.

Cody80.
William J. Cody, Jr. and William Waite, *Software Manual for the Elementary Functions,* Computational Mathematics, Prentice Hall, Englewood Cliffs, NJ, 1980.

Comm81.
IEEE 754 Committee, "A Proposed Standard for Binary Floating-Point Arithmetic," *Computer,* vol. 14, no. 3, pp. 51-62, IEEE Computer Society, March 1981.

Feue82.
Alan R. Feuer, *The C Puzzle Book,* Prentice Hall, Englewood Cliffs, NJ, 1982.

Feue84.
Alan R. Feuer and Narain Gehani, *Comparing and Assessing Programming Languages,* Prentice Hall, Englewood Cliffs, NJ, 1984.

Geha85.
Narain Gehani, *Advanced C: Food for the Educated Palate,* Computer Science Press, Rockville, MD, 1985.

Harb84.
Samuel Harbison and Guy Steele, Jr., *C: A Reference Manual,* Prentice-Hall, Englewood Cliffs, NJ, 1984.

Kern78.
Brian W. Kernighan and Dennis M. Ritchie, *The C Programming Language,* Prentice Hall, Englewood Cliffs, NJ, 1978.

Knut81.
Donald E. Knuth, *Seminumerical Algorithms,* The Art of Computer Programming, 2, Addison-Wesley, Reading, MA, 1981. Second Edition.

Past85.
Irene Pasternack, *Exploring the UNIX Environment,* Bantam Books, NYC, NY, 1985.

Spaf83.
Eugene H. Spafford, A Report on the Accuracy of Prime Computers Floating Point Software and Hardware, Georgia Institute of Technology, Atlanta, GA, 1983. Technical Report GIT-ICS-83/09.

For Additional Copies

Copies of this technical report may be requested by sending US mail to the authors at the address on the title page or from

{akgua,decvax,hplabs,ihnp4,seismo}!gatech!tech-reports

Local User Groups

The USENIX Association will support local user groups in the United States and Canada in the following ways:

- Assisting the formation of a local user group by doing an initial mailing for the group. This mailing may consist of a list supplied by the group, or may be derived from the USENIX membership list for the geographical area involved. At least one member of the organizing group must be a current member of the USENIX Association. Membership in the group must be open to the public.
- Publishing information on local user groups in *;login:* giving the name, address, phone number, net address, time and location of meetings, etc. Announcements of special events are welcome; send them to the editor at the USENIX office.

Please contact the USENIX office if you need assistance in either of the above matters. Our current list of local groups follows.

In the **Boulder**, Colorado area a group meets about every two months at different sites for informal discussions.

Front Range Users Group
N.B.I., Inc.
P.O. Box 9001
Boulder, CO 80301

Steve Gaede (303) 444-5710
hao!nbires!gaede

Dallas/Fort Worth UNIX User's Group
Seny Systems, Inc.
5327 N. Central, #320
Dallas, TX 75205

Jim Hummel (214) 522-2324

In the **Washington, D.C.,** area there is an umbrella organization called Capitol Shell. It consists of commercial, government, educational, and individual UNIX enthusiasts. For information and a newsletter write:

Capitol Shell
8375 Leesburg Pike, #277
Vienna, VA 22180

seismo!cal-unix!capish

In the **New York City** area there is a non-profit organization for users and vendors of products and services for UNIX systems.

Unigroup of New York
G.P.O. Box 1931
New York, NY 10116

In **Minnesota** a group meets on the first Wednesday of each month. For information contact:

UNIX Users of Minnesota
Carolyn Downey (612) 934-1199

In the **Atlanta** area there is a group for people with interest in UNIX or UNIX-like systems:

Atlanta UNIX Users Group
P.O. Box 12241
Atlanta, GA 30355-2241

Marc Merlin (404) 255-2848
Mark Landry (404) 874 6037

In the **Seattle** area there is a group with over 150 members, a monthly newsletter and meetings the fourth Tuesday of each month.

Seattle/UNIX Group
P.O. 58852
Seattle, WA 98188

Irene Pasternack (206) FOR-UNIX
uw-beaver!tikal!ssc!slug

An informal group is starting in the **St. Louis** area:

St. Louis UNIX Users Group
Plus Five Computer Services
765 Westwood, 10A
Clayton, MO 63105

Eric Kiebler (314) 725-9492
ihnp4!plus5!sluug

In the **northern New England** area is a group that meets monthly at different sites. Contact one of the following for information:

Emily Bryant (603) 646-2999
Kiewit Computation Center
Dartmouth College
Hanover, NH 03755

decvax!dartvax!emilyb

David Marston (603) 883-3556
Daniel Webster College
University Drive
Nashua, NH 03063

A UNIX/C language users group has been formed in **Tulsa**. For current information on meetings, etc. contact:

Pete Rourke
$USR
7340 East 25th Place
Tulsa, OK 74129

The **New Zealand** group provides an annual Workshop and Exhibition and a regular newsletter to its members.

New Zealand UNIX Systems User Group
P.O. Box 13056
University of Waikato
Hamilton, New Zealand

Publications Available

The following publications are available from the Association Office or the source indicated. Prices and overseas postage charges are per copy. California residents please add applicable sales tax. Payments **must** be enclosed with the order and **must** be in US dollars payable on a US bank.

USENIX Conference Proceedings

Meeting	Location	Date	Price	Overseas Mail Air	Overseas Mail Surface	Source
USENIX	Denver	Winter '86	$20	$25	$5	USENIX
USENIX	Portland	Summer '85	$25	$25	$5	USENIX
USENIX	Dallas	Winter '85	$20	$25	$5	USENIX
USENIX	Salt Lake	Summer '84	$25	$25	$5	USENIX
UniForum	Wash. DC	Winter '84	$30	$20		/usr/group
UNICOM	San Diego	Winter '83		*sold out*		

USENIX Association
P.O. Box 7
El Cerrito, CA 94530

/usr/group
4655 Old Ironsides Dr., #200
Santa Clara, CA 95050

EUUG Publications

The following EUUG publications may be ordered from the USENIX Association office.

The EUUG Newsletter, which is published four times a year, is available for $4 per copy or $16 for a full-year subscription. The earliest issue available is Volume 3, Number 4 (Winter 1983).

The July 1983 edition of the EUUG Micros Catalog is available for $8 per copy.

4.2BSD Manual Reproduction Authorization and Order Form

USENIX Member No.: _____

Purchase Order No.: _____

Date: _____

As the representative of a USENIX Association Institutional or Supporting Member in good standing, and as a *bona fide* license holder of both a 4.2BSD software license from the Regents of the University of California and a UNIX™/32V or System III or System V license or sublicense from Western Electric Company, and pursuant to the copyright notice as found on the rear of the cover page of the UNIX/32V Programmer's Manual stating that

> "Holders of a UNIX™/32V software license are permitted to copy this document, or any portion of it, as necessary for licensed use of the software, provided this copyright notice and statement of permission are included",

I hereby appoint the USENIX Association as my agent, to act on my behalf to duplicate and provide me with such copies of the Berkeley 4.2BSD Manuals as I may request.

Signed: _____

Institution: _____

Ship to:

Name: _____

_____

_____

_____

_____

Phone: _____

Billing address, if different:

Name: _____

_____

_____

_____

_____

Phone: _____

The prices below **do not** include shipping and handling charges or state or local taxes. All payments must be in US dollars drawn on a US bank.

4.2BSD User's Manual	_____	at $18.50 each =	$_____
4.2BSD Programmer's Manual	_____	at $17.00 each =	$_____
4.2BSD System Manager's Manual	_____	at $ 9.50 each =	$_____
Total	_____		$_____

[ ] Purchase order enclosed; invoice required
(Purchase orders **must** be enclosed with this order form.)

[ ] Check enclosed for the manuals: $_____
(Howard Press will send an invoice for the shipping and handling charges and applicable taxes.)

Make your check or purchase order out to Howard Press and mail it with this order form to:

Howard Press
c/o USENIX Association
P.O. Box 7
El Cerrito, CA 94530

for office use: l.v.: _____ check no.: _____ amt. rec'd: _____

USENIX Association
P.O. Box 7
El Cerrito, CA 94530

First Class Mail

FIRST CLASS MAIl
U.S. POSTAGE PAIl
El Cerrito, CA 9453(
Permit No. 87

Summer 1986 Conference at Atlanta

Accuracy of Some Floating Point Math Functions on Selected UNIX Computers

First 1986 Distribution Tape

Elections for the USENIX Officers and Board Candidates' Position Statements

PLEASE VOTE

Change of Address Form

Please fill out and send the following form through the U.S. mail to the Association Office at the address above.

Name: _____ Member #: _____

OLD: _____ NEW: _____

_____ _____

_____ _____

_____ _____

Phone: _____

uucp: {decvax,ucbvax}! _____